CS 360 with Mohammad Hajiabadi

Eason Li

2025 F

Contents

1 Finite State (Memory) Automata (Machines)

Lecture 1 - Wednesday, September 03

Definition 1.1: String

A string over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ of symbols from Σ.

Definition 1.2: Empty String

The empty string is the string with no symbols and is denoted ε.

Comment 1.1

The set of all strings, including the empty string, over an alphabet Σ is denoted Σ^*.

Definition 1.3: Finite Automaton

A **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the *states*,
2. Σ is a finite set called the *alphabet*,
3. $\delta : Q \times \Sigma \to Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the *set of accept states*.

Definition 1.4: Accept

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M **accepts** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}, \quad$ for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Definition 1.5: Regular Language

A language is called a **regular language** if some finite automaton recognizes it.

Definition 1.6: Nondeterministic Finite Automaton

A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function,

4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

1.1 What languages can or cannot be accepted by DFAs? Or what is the power of DFAs?

Lecture 2 - Monday, September 08

In the future, we want to answer the following question: What languages can/cannot be accepted by normal computers (i.e., algorithms that output 1 on the strings in the language and output 0 otherwise.)

Definition 1.7: Equivalent

Two DFAs/NFAs M_1, M_2 are **equivalent** if $L(M_1) = L(M_2)$, i.e., the language determined by the two machines are the same.

Theorem 1.1

For every NFA N, there exists an equivalent DFA D. Hence, a language is accepted by an NFA if and only if it is accepted by a DFA.

Proof. Suppose that N has no ε-transitions. Given the NFA $N = (Q, \Sigma, \delta, 0, F)$ with $Q = \{0, \ldots, n-1\}$, we construct a DFA $D = (Q_D, \Sigma, \delta_D, q_0', F_D)$. The alphabet remains unchanged.

First, we define the set of states of D as

$$Q_D \overset{\text{def}}{=} \{q_S \mid S \subseteq \{0, \ldots, n-1\}\}$$

Next, we define the start state of D as $q_0' \overset{\text{def}}{=} q_{\{0\}}$.

The main idea is that after reading the input $w_1 \ldots w_\ell$, the DFA D will end in state q_S, where S is precisely the set of all states in $\{0, \ldots, n-1\}$ that are reachable in N after processing $w_1 \ldots w_\ell$.

The transition function of D is then defined by

$$\delta_D(q_S, a) = q_{S'}, \quad \text{where } S' = \bigcup_{p \in S} \delta(p, a)$$

Finally, the set of accept states F_D consists of all states q_S such that $S \cap F \neq \varnothing$.

As a concluding question, if $|F| = t$, what can we say about the value of $|F_D|$? ($(2^t - 1)2^{n-t}$).

Now we consider having the ε-transitions: For any set of states $S \subseteq \{0, \ldots, n-1\}$, let $E(S)$ contain all elements of S as well as those that can be reached from S by making one or more ε transitions. Now we modify δ_D:

$$\delta_D(q_s, a) = q_{S'}$$

where $S' = \bigcup_{p \in S} E(\delta(p, a)) = E\Big(\bigcup_{p \in S} \delta(p, a)\Big)$. □

Question 1.1.

Can you give a DFA with the same number of states as of the underlying NFA (in general)?

Solution. No. □

Proposition 1.1

If L is regular, $\overline{L} = \{\text{binary string } x : x \notin L\}$ is regular.

Proof. If $M = (Q, \Sigma, \delta, q_0, F)$ is a FA with $L(M) = L$. Then define

$$M' = (Q, \Sigma, \delta, q_0, F') \text{ as a FA, where } F' = Q \setminus F$$

It follows that $L(M') = \overline{L}$. □

Definition 1.8: Union, Concatenation, (Kleene) Star

Let A and B be languages. We define the regular operations **union**, **concatenation**, and **star** as follows:

- **Union:** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation:** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star:** $A^* = \{x_1x_2 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

1.2 The class of regular languages is closed under the union, concatenation, and star operation.

Theorem 1.2

The class of regular languages is closed under the union operation.

Theorem 1.3

The class of regular languages is closed under the concatenation operation.

Theorem 1.4

The class of regular languages is closed under the star operation.

Question 1.2.

Can you show that the language $L = \{a^\ell b^\ell : \ell \geq 0\}$ is not a regular language?

Proof. Contradiction via pumping lemma. □

1.2.1 Pumping Lemma

Theorem 1.5: Pumping Lemma

Let L be a regular language. Then there exists an integer $p \geq 1$ depending only on L such that every string w in L of length at least p (p is called the "pumping length") can be written as $w = xyz$ (i.e., w can be divided into three substrings), satisfying the following conditions:

1. $|y| \geq 1$
2. $|xy| \leq p$
3. $(\forall n \geq 0)(xy^n z \in L)$

Lecture 3 - Wednesday, September 10

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite-state automaton that accepts the regular language L. Let $p := |Q|$, and let $w = a_1 a_2 \cdots a_n \in L$ be a word of length $n \geq p$. Define a sequence of states $q_0, q_1, \ldots, q_n$ inductively by

$$q_i = \delta(q_{i-1}, a_i) \quad \text{for } i = 1, \ldots, n$$

Since the sequence $q_0, \ldots, q_p$ has length $p+1$ and since there are only p states in Q, by the pigeonhole principle there exist indices $0 \leq i < j \leq p$ such that $q_i = q_j$. Now define:

$$x = a_1 a_2 \cdots a_i, \quad y = a_{i+1} a_{i+2} \cdots a_j, \quad z = a_{j+1} \cdots a_n$$

Then $w = xyz$, $|xy| \leq p$, and $|y| \geq 1$. Moreover, since $q_i = q_j$, reading y causes the machine to loop back to q_i, so for all $k \geq 0$,

$$\delta(q_0, xy^k z) = \delta(q_i, y^k z) = \delta(q_i, z)$$

But $\delta(q_i, z) = q_n \in F$, since $w \in L$. Hence, $xy^k z \in L$ for all $k \geq 0$, and we obtain the desired result. □

Exercise 1.1

Show that the language $L = \{0^k : k \text{ is composite}\}$ is not regular.

Proof. Show that the language $L' = \{0^k : k \text{ is prime}\}$ is not regular, and use the fact that the complement of a regular language is regular to show that L is not regular either. □

Exercise 1.2

Show whether the language $L = \{\text{binary strings with equal number of 01 and 10 substrings}\}$ is regular.

1.3 Regular Expression

Definition 1.9: Regular Expression

R is a regular expression over an alphabet Σ if R is:

1. ε or $\varnothing$ or a for some $a \in \Sigma$.
2. $(R_1 \cup R_2)$ or $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions.
3. (R^*), where R is a regular expression.

Theorem 1.6: Precedence Order

Star, then concatenation, then union.

Comment 1.2

If R is a regular language, then

$$R^+ := RR^*$$

1.3.1 Some Examples and Practices

Example 1.1

For a set $A = \{a_1, \ldots, a_n\}$ we sometimes use A as a regular expression to mean $A := a_1 \cup \cdots \cup a_n$.

Example 1.2

Strings over Σ of even length can be expressed as $(\Sigma\Sigma)^*$.

Exercise 1.3

Is it true that $(0^*1^*)^* = \{0,1\}^*$?

Solution. Yes, because $(0^*1^*)^* = (0^*1^*)(0^*1^*)\cdots(0^*1^*)$. □

Exercise 1.4

What is the regular expression for strings not ending with 11.

Solution. $\varepsilon \cup \{0,1\} \cup \{0,1\}^*(00 \cup 01 \cup 10)$. □

Exercise 1.5

What is the regular expression for strings who has exactly one 1 and an even number of 0s.

Solution. $(00)^*1(00)^* \cup (00)^*010(00)^*$. □

Exercise 1.6

Find the regular expression for strings containing at most one pair of consecutive 0s.

Solution. $(1 \cup 01)^*(\varepsilon \cup 0 \cup 00)(1 \cup 10)^*$. □

Theorem 1.7

A language is regular if and only if some regular expression describes it.

Proof. [$\Longleftarrow$]: Easy.
[$\Longrightarrow$]: Stronger result to prove: if A is recognized by a generalized finite state automaton, then A is also described by a regular expression. A generalized finite state automaton is can be transformed such that

1. The start state has transition arrows going to every other state but no arrows coming in from any other state.
2. There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
3. Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

Then we can get rid of all states $q_{rip} \notin \{q_0, q_{accept}\}$ one by one. □

2 Context Free Grammars and Languages

Lecture 4 - Monday, September 15

Definition 2.1: Context-Free Grammar

A **context-free grammar** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V, called the **terminals**,
3. R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Let A be a variable and $A \to \gamma$ be a rule in R. Then for $\alpha, \beta \in (V \cup \Sigma)^*$ we write $\alpha A \beta \Rightarrow_G \alpha\gamma\beta$ and say $\alpha\gamma\beta$ is derivable from $\alpha A \beta$ in one step. A sequence of the form $S \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n$ is called a derivation in n steps. We let $\Rightarrow_G^*$ denote derivability in zero or more steps. Language of G is

$$L(G) = \{x \in \Sigma^* : S \Rightarrow_G^* x\}$$

Definition 2.2: Context-Free Language

A language R is context-free if $R = L(G)$ for some CFG G.

2.1 Example CFGs

2.1.1 CFG for $A = \{x : x$ has same number of of 0s and 1s$\}$

Grammar $G = (\{S\}, \{0,1\}, P, S)$ with P being:

$$S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon$$

Proof. [$L(G) \subseteq A$]: suppose $x \in L(G)$; we show $x \in A$ by induction on the length ℓ of the shortest deriving path for x. Base: $\ell = 1$: then $x = \varepsilon$, and so $x \in A$.

$\ell \geq 2$: consider all possible "first moves":

1. $S \Rightarrow_G 0S1 \Rightarrow_G^* 0y1$ and $x = 0y1$. We have $S \Rightarrow_G^* y$ in $\ell - 1$ steps, and so $y \in A$, and hence $x \in A$.
2. $S \Rightarrow_G 1S0 \Rightarrow_G^* 1y0$ and $x = 1y0$. Like above, $y \in A$, and hence $x \in A$.
3. $S \Rightarrow_G SS \Rightarrow_G^* yz$ and $x = yz$. Then, $S \Rightarrow_G^* y$ and $S \Rightarrow_G^* z$, each derivable in at most $\ell - 1$ steps. Thus, $y \in A$ and $z \in A$, and hence $x \in A$.

Question 2.1. Why not induction on $|x|$?

Case 3 will go wrong: it might be that, e.g., $y = \varepsilon$ and hence $x = z$, so we can't inductively argue that $z \in A$.

[$A \subseteq L(G)$]. Suppose $x \in A$. We prove $x \in L(G)$ by induction on $\ell = |x|$. Base: $|x| = 0$: then $x = \varepsilon$, and so $x \in L(G)$.

$|x| = 2(k+1)$ and $k \geq 0$. Then:

1. $x = 0y1$: then $y \in A$ and $|y| = 2k$, and by induction $y \in L(G)$, and so $x \in L(G)$.
2. $x = 1y0$: then $y \in A$ and $|y| = 2k$, and hence $x \in L(G)$, like above.
3. $x = 0y0$ or $x = 1z1$. In this case we can write $x = x'x''$ such that $x', x'' \in A$ (Why?) and $|x'| \geq 2$, $|x''| \geq 2$. By induction, $x', x'' \in L(G)$, and so $x \in L(G)$.

Why (proof)? Suppose $x = 0y0$. For $1 \leq i \leq |x|$, let

$$\Delta_i = \#0\text{'s} - \#1\text{'s in } x[i] \overset{\text{def}}{=} x_1 \cdots x_i$$

We have $\Delta_1 = 1$, $\Delta_{|x|-1} = -1$ and for all $1 \leq i < |x|$: $|\Delta_i - \Delta_{i+1}| = 1$. Thus, for some $1 \leq i \leq |x| - 1$: $\Delta_i = 0$, and hence $x[i] \in A$.

Similar proof if $x = 1y1$. □

2.1.2 CFG for Palindromes: $A = \{w \in \{a,b\}^* : w^R = w\}$

Grammar $G = (\{S\}, \{0,1\}, P, S)$ with P being:

$$S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

Proof. [$L(G) \subseteq A$]. Let $x \in L(G)$. We show $x \in A$ by induction on the length ℓ of the shortest deriving path for x.

$\ell = 1$, then $x = \varepsilon$, $x = 0$ or $x = 1$. Thus, $x \in A$.

$\ell \geq 2$, consider all possible first moves:

1. $S \Rightarrow_G 0S0 \Rightarrow_G^* 0y0$ and $x = 0y0$. We have $S \Rightarrow_G^* y$ in $\ell - 1$ steps, and so $y \in A$, and hence $x \in A$.
2. $S \Rightarrow_G 1S1 \Rightarrow_G^* 1y1$ and $x = 1y1$. We have $S \Rightarrow_G^* y$ in $\ell - 1$ steps, and so $y \in A$, and hence $x \in A$.

[$A \subseteq L(G)$]. Suppose $x \in A$. We prove $x \in L(G)$ by induction on $|x|$.

1. Base: $|x| = 0$ or $|x| = 1$: then $x = \varepsilon$, $x = 0$ or $x = 1$, and so $x \in L(G)$.
2. If $|x| \geq 2$: then either $x = 0y0$ or $x = 1y1$ for some palindrome y. By the induction hypothesis, $y \in L(G)$, namely $S \Rightarrow_G^* y$.
   (a) If $x = 0y0$, then $S \Rightarrow_G 0S0 \Rightarrow_G^* 0y0$, and hence $x \in L(G)$.
   (b) If $x = 1y1$, similarly to (a).

Question 2.2. Why not the base induction only over $x = \varepsilon$ (i.e., $|x| = 0$)?

Item (2) assumes $|x| \geq 2$, and so doesn't apply to $x = 0$ or $x = 1$.

□

2.1.3 CFGs for $A = \{w \in \{(,)\}^* : w$ is balanced$\}$

Comment 2.1

Formal definition: Map "(" $\mapsto 0$ and ")" $\mapsto 1$. For w, let $w[i] = w_1 \cdots w_i$. Let $\Delta_i(w) = \#0\text{s} - \#1\text{s}$ in $w[i]$. Then

$$A = \{\, w \in \{0,1\}^* \mid w \text{ has an equal number of 0s and 1s, and } \forall i \leq |x| - 1 : \; \Delta_i(w) \geq 0 \,\}$$

Grammar $G_1 : S \to (S) \mid (SS) \mid SS \mid \varepsilon$. Let the underlying language be L_1.
Grammar $G_2 : S \to (S)S \mid \varepsilon$. Let the underlying language be L_2.

Exercise 2.1

Show that $L_1 = L_2 = A$.

2.2 Parse Tree

A parse tree is a way of representing derivations which ignores the order followed by variables when they are expanded.

Definition 2.3: Parse Tree

Let $G = (V, \Sigma, P, S)$ be a CFG. A tree T is a **parse tree** for G if

1. each internal node of T are labeled by a variable.
2. each leaf is labeled by a **terminal** or ε. If a leaf is labeled ε, then it is the only child of its parent.
3. if an internal node is labelled by A, and its children are labeled (in order) by $X_1, \ldots, X_k$, then $A \to X_1 \ldots X_k$ is a rule in P.

2.2.1 Yield of a Parse Tree

Definition 2.4: Yield of a Parse Tree

The **yield of a parse tree** is the concatenation of all terminals in the leaves, from left to right.

Comment 2.2

The yield is always in the language of the grammar.

2.3 Ambiguous vs Non-Ambiguous Grammar

Definition 2.5: Leftmost Derivation

The **leftmost derivation** is the derivation in which the leftmost variable in the currently derived string is expanded.

Theorem 2.1

For each grammar $G = (V, \Sigma, P, S)$ and string $w \in \Sigma^*$, w has two distinct parse trees iff w has two distinct leftmost derivations from S.

Definition 2.6: Ambiguous

A string w is derived **ambiguously** in context-free grammar G if it has two or more different leftmost derivations. Grammar G is **ambiguous** if it generates some string ambiguously.

Lecture 5 - Wednesday, September 17

2.4 Power and Limitations of CFGs

2.4.1 Pumping Lemma (for CFL)

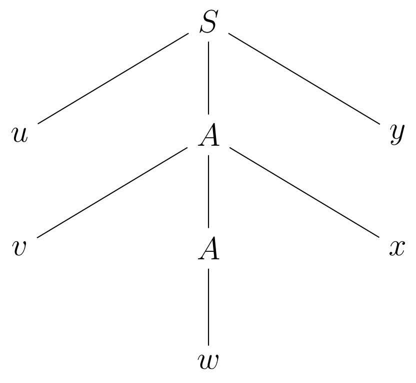
Theorem 2.2: Pumping Lemma

If L is a CFL there is a number p such that for all $s \in L$ with $|s| \geq p$ we have that

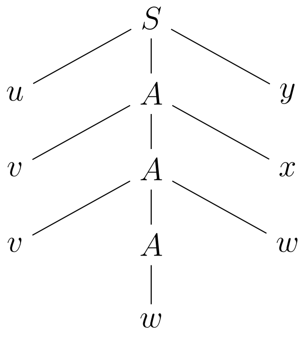
1. $s = uvwxy$
2. $|vx| > 0$
3. $|vwx| \leq p$
4. $uv^nwx^ny \in L$ for all $n \geq 0$

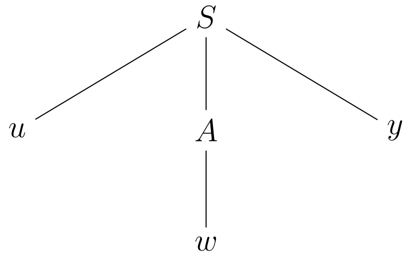
Proof. If L is a CFL, there is some grammar G for L. The arity of G is the length k of the longest right-hand side in G. Let n be number of variables in G. Consider a string s in L with length greater than or equal to $p = k^{n+1}$. The parse tree for this string has a height greater than or equal to $n + 1$. This means that some path from the root down to a leaf in the parse tree has at least $n + 1$ internal nodes on the path. This in turn means that at least one variable along that path has to be repeated. Let A be the first variable on that path to get repeated as we work our way up from the leaf to the root along that path.

This results in a parse tree that contains structures like this:



If this is a valid parse tree, then these are also valid parse trees for the string uv^2wx^2y and uwy respectively:





□

2.4.2 $A = \{0^i1^i2^i : i \geq 0\}$ is not a CFL

Proof. Let p be as in the pumping lemma. Define $s = 0^p1^p2^p$. By pumping lemma, we may write $\alpha = uvxyz$ for some x, y, z such that

1. $|vxy| \leq p$; and
2. $|vy| \geq 1$; and
3. $uv^ixy^iz \in E$ for all $i \geq 0$.

However, it is impossible: vxy cannot contain all the three symbols $\{0, 1, 2\}$, and so for some $b \in \{0, 1, 2\}$, the pumped-up string uv^2xy^2z will have less b symbols than some symbol in $\{0, 1, 2\} - \{b\}$. □

2.4.3 $A = \{ww : w \in \{0, 1\}^*\}$ is not a CFL

Proof. Let p be as in the pumping lemma. Define $s = 0^p1^p0^p1^p$ and note $|s| \geq p$. By pumping lemma, we may write $s = uvxyz$ for some x, y, z such that

1. $|vxy| \leq p$; and
2. $|vy| \geq 1$; and
3. $uv^ixy^iz \in A$ for all $i \geq 0$.

Now we have three cases:

- If vxy is entirely contained in first half, then the second half of $uv^2xy^2z = 0^{p+k}1^{p+f}0^p1^p$ where $k + f \leq p$. Thus, the second half starts with 1 but the first half with 0, so it isn't in A.
- If vxy is entirely contained in second half, same as above.
- If vxy contains some symbols from first half and some from second half, then setting $i = 0$, we get $0^p1^k0^t1^p$, where either $k < p$ or $t < p$.

Either case, it is not in A. □

2.5 Closure Properties of CFLs

2.5.1 CFLs are closed under Union, Concatenation, and Star

For $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$, let $S_{start} \notin V_1 \cup V_2$, we have

1. `Union`: the union is

$$(S_{start} \cup V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S_{start} \to S_1\} \cup \{S_{start} \to S_2\}, S_{start})$$

2. `Concatenation`, the concatenation is

$$(S_{start} \cup V_1 \cup V_2, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S_{start} \to S_1 S_2\}, S_{start})$$

3. `Star`: the star $L(G)^*$ is

$$(S_{start} \cup V_1, \Sigma_1, R_1 \cup \{S_{start} \to S_1 S_{start} \mid \varepsilon\}, S_{start})$$

2.5.2 CFLs are NOT closed under Intersection

We know that $A = \{0^i 1^i 2^j : i, j \geq 0\}$ is context free. Similarly, $B = \{0^j 1^i 2^j : i, j \geq 0\}$ is also context free. We know that $A \cap B = \{0^i 1^i 2^i : i, j \geq 0\}$ isnt context free, serving as a counterexample.

2.5.3 CFLs are NOT closed under Complementation

2.6 CFLs are more powerful than regular languages

Lecture 6 - Monday, September 22

2.6.1 Regular Expressions to CFGs

$\varnothing$ can be generated by a CFG with no production rules. a and ε can be generated by a CFG $S \to a$ and by $S \to \varepsilon$, respectively. Now suppose R_1 is generated by (V_1, Σ, P_1, S_1) and R_2 is generated by (V_2, Σ, P_2, S_2), where $V_1 \cap V_2 = \varnothing$.

- $R_1 \cup R_2$ can be generated by $\big(V_1 \cup V_2 \cup \{S_{\text{new}}\},\ \Sigma,\ P_1 \cup P_2 \cup \{S_{\text{new}} \to S_1 \mid S_2\},\ S_{\text{new}}\big)$, where $S_{\text{new}} \notin V_1 \cup V_2$.
- $R_1 \cdot R_2$ can be generated by $\big(V_1 \cup V_2 \cup \{S_{\text{new}}\},\ \Sigma,\ P_1 \cup P_2 \cup \{S_{\text{new}} \to S_1 S_2\},\ S_{\text{new}}\big)$, where $S_{\text{new}} \notin V_1 \cup V_2$.
- R_1^* can be generated by $\big(V_1 \cup \{S_{\text{new}}\},\ \Sigma,\ P_1 \cup \{S_{\text{new}} \to \epsilon \mid S_1 S_{\text{new}}\},\ S_{\text{new}}\big)$, where $S_{\text{new}} \notin V_1$.

2.6.2 DFAs to CFGs

We start with a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

1. Make a variable R_i for each $q_i \in Q$.
2. Add the rule $R_i \to aR_j$ if $\delta(q_i, a) = q_j$.
3. Add the rule $R_i \to \epsilon$ if $q_i \in F$.
4. The start variable is R_0.

To prove that this is correct, we use induction, which is left as an exercise.

2.6.3 Algorithmic Aspects of DFAs/CFLs

Question 2.3. Given a DFA M, can we check if $L(M) = \emptyset$?

Yes. Check if an accepting state is reachable from the start state.

Question 2.4. Given a CFG G, can we check if $L(G) = \emptyset$?

Algorithm: Let $Y = \emptyset$. While there exists $A \to \gamma$, such that γ has only terminals and/or variables in Y, add A to Y. Check if start variable is in Y.

Question 2.5. Given two DFAs M_1, M_2, can we check if $L(M_2) = L(M_2)$?

We have $L(M_1) = L(M_2)$ iff both $L(M_1) \cap \overline{L(M_2)} = \varnothing$ and $\overline{L(M_1)} \cap L(M_2) = \varnothing$. Use Questino 2.3 answer to check if $L(M_1) \cap \overline{L(M_2)} = \varnothing$ and $\overline{L(M_1)} \cap L(M_2) = \varnothing$.

Question 2.6. Given two CFGs G_1, G_2, can we check if $L(G_2) = L(G_2)$?

Comment 2.3

We will prove this later.

2.6.4 Intersection of Regular Languages and CLFs

We want a PDA that accepts $L(M) \cap L(P)$, where

- $M = (Q_M, \Sigma, \delta_M, q_0^M, F_M)$ is a DFA, and
- $P = (Q_P, \Sigma, \Gamma, \delta_P, q_0^P, Z_0, F_P)$ is a PDA.

We define the new PDA P' as follows:

- **States:** pairs of a DFA state and a PDA state:

$$Q' = Q_M \times Q_P$$

- **Start state:** (q_0^M, q_0^P).
- **Stack alphabet:** the same as P, i.e. Γ.
- **Transition function:**
  When M reads a symbol $a \in \Sigma$ and P makes a transition on a with stack action, we combine them: If

$$\delta_M(q_M, a) = q'_M$$

$$\delta_P(q_P, a, X) \ni (q'_P, \gamma),$$

  then in P' we define

$$\delta'((q_M, q_P), a, X) \ni \big((q'_M, q'_P), \gamma\big).$$

  That is, M's state updates deterministically, while P's state and stack update as before.

- **Accepting states:**

$$F' = F_M \times F_P.$$

3 Push Down Automata

Imagine a FA with an unlimited extra "stack" memory. When reading something from the input we may also look at the top element in the stack, and we can pull that out or push something new.

Example 3.1: $A = \{0^n 1^n : n \geq 0\}$

Start state q_0 and stack initialized to be empty. Start by putting \$ on the stack without reading any input symbol. Keep reading 0s and push 0 into the stack. When hitting a 1 from the input, pop 0 from the stack for any 1 read from the input. When hitting \$ on the stack go to accept state.

3.1 PDA Definition

Definition 3.1: PDA

A **PDA** is a 6-tuple $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where

$$\begin{array}{rcl rcl} Q &=& \text{finite set of states;} & q_0 &=& \text{initial state;} \\ \Sigma &=& \text{alphabet of input symbols;} & F &=& \text{accept states;} \\ \Gamma &=& \text{alphabet of stack symbols;} & \delta &=& \text{transition function} \end{array}$$

where the transitions is defined as

$$\delta(q, a, \alpha) = (q', \beta) \quad \text{for } q, q' \in Q, \;\; a \in \Sigma \cup \{\varepsilon\}, \;\; \alpha, \beta \in \Gamma \cup \{\varepsilon\}$$

to mean that if $\alpha \in \Gamma$: if the top symbol is α, pop it and push β; If $\alpha = \varepsilon$, read nothing from stack and push β.

Note 3.1

ε appears as an option in three places:

- Not reading an input symbol.
- Not reading (popping) a stack symbol.
- Not pushing a stack symbol on the stack.

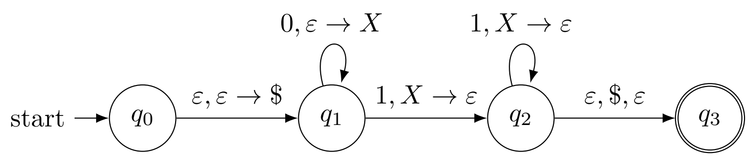
3.1.1 Example PDAs

Example: $A = \{0^n 1^n : n \geq 1\}$

Example 3.2

For $A = \{0^n 1^n : n \geq 1\}$, the corresponding PDA is

$$P = (Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}, \Gamma = \{\$, X\}, \delta, \{q_0\}, \{q_3\})$$



Example: $A = \{0^n 1^n : n \geq 0\}$

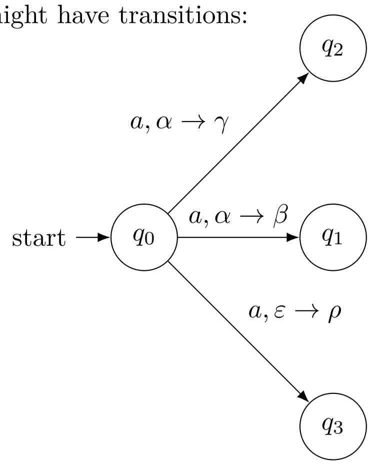
Similar as above, the only difference is to

replace $(1, X \to \varepsilon)$ between q_1 and q_2 with $(\varepsilon, \varepsilon \to \varepsilon)$.

3.2 PDAs are Non-Deterministic

Example 3.3

PDAs are non-deterministic: We might have transitions:

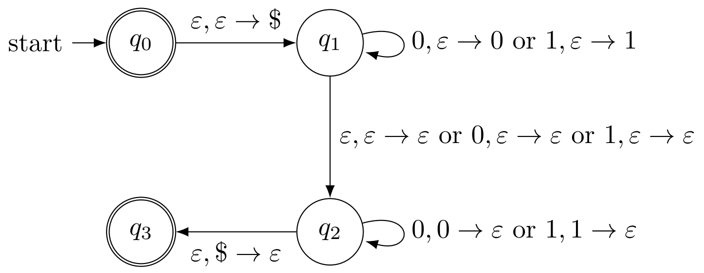


3.2.1 Example PDAs

Example: Palindromes $\{w \in \{0,1\}^* : w = w^R\}$

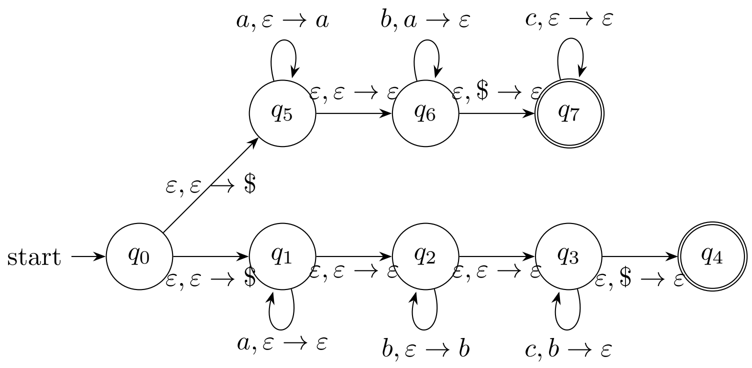
Example 3.4

For $\{w \in \{0,1\}^* : w = w^R\}$, we have the following PDA:


start
q_0
$\varepsilon, \varepsilon \to \$$
q_1
$0, \varepsilon \to 0$ or $1, \varepsilon \to 1$
$\varepsilon, \varepsilon \to \varepsilon$ or $0, \varepsilon \to \varepsilon$ or $1, \varepsilon \to \varepsilon$
q_3
q_2
$0, 0 \to \varepsilon$ or $1, 1 \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$

Example: $L = \{a^i b^i c^j : i, j \geq 0\} \cup \{a^i b^j c^j : i, j \geq 0\}$

Example 3.5

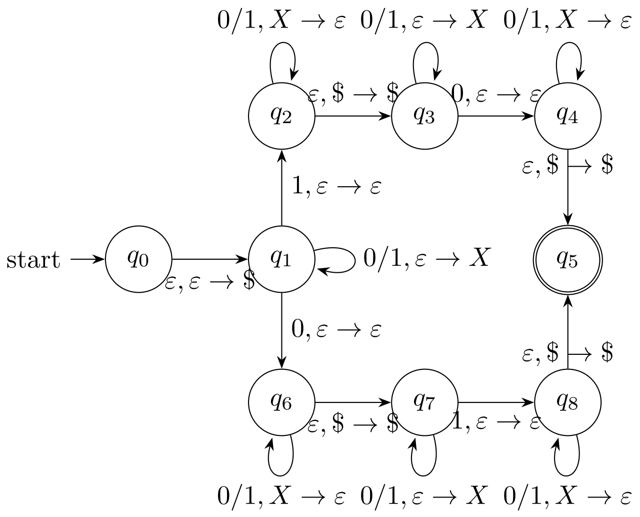

start
q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
$a, \varepsilon \to a$
$b, a \to \varepsilon$
$c, \varepsilon \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$a, \varepsilon \to \varepsilon$
$b, \varepsilon \to b$
$c, b \to \varepsilon$

Lecture 7 - Wednesday, September 24

Example: $A = \{x \in \{0,1\}^* : x$ **is not of the form** $ww\}$

Example 3.6

If $|x|$ is odd, then $x \in A$, so we just do it for the case $|x|$ is even and $|x| \geq 2$. (We can union two PDAs easily.) In the case when $|x|$ is even, it is in A if $x = uv$ such that $u = u_1 a u_2$, $v = v_1 b v_2$, and $|u_1 v_2| = |v_1 u_2|$ and $a \neq b$. The PDA guesses the positions of a and b, checks if $a \neq b$. The PDA is:



3.3 PDAs and CFGs have the same power

Theorem 3.1

The class of languages accepted by PDAs is exactly CFL.

Proof. We first show how to build a PDA from a CFL.

1. $\delta(q_{start}, \epsilon, \epsilon) = \{(q_{loop}, S\$)\}$ $\{$Place \$ and S on the stack$\}$
2. $\delta(q_{loop}, \epsilon, A) = \{(q_{loop}, u) : A \to u$ is a rule of $G\}$ $\{$select a rule with A on LHS and push RHS onto stack$\}$
3. $\delta(q_{loop}, a, a) = \{(q_{loop}, \epsilon)\}$. $\{$match terminal symbol in input to one in rule$\}$
4. $\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \epsilon)\}$ $\{$accept if stack empty and input read$\}$

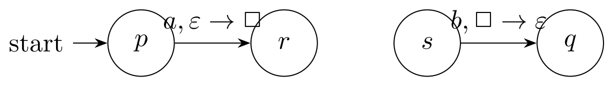
Now we show PDAs P to CFGs G. We assume P is of the following form:

1. Has a single accept state q_{accept}
2. Empties its stack before accepting
3. Either pushes a symbol onto the stack or pops one off, but not both in one move.

Suppose $P = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept})$.

- For every $p, q \in Q$, add a variable A_{pq} which is to generate all strings which take p with empty stack to q with empty stack.
- Start variable: $A_{q_0 q_{accept}}$.
- Add $A_{pp} \to \epsilon$ to the production rules for all $p \in Q$.
- Adding productions rules whose lefthand side is A_{pq} for all $p, q \in Q$ (including the case $p = q$)
- Idea: Any x that takes p with empty stack to q with empty stack, the first move is a push and the last is a pop. The last symbol popped is either the first symbol (first case), or not (second case). We will add rules to account for both first case and second case.

`[Case one]`: If for $r, s \in Q$ and $a, b \in \Sigma \cup \{\varepsilon\}$:



We add

$$A_{pq} \to a A_{rs} b$$

Interpretation: if a string y takes r with empty stack to s with empty stack, then ayb takes p with empty stack to q with empty stack.

`[Case two]`:

$$\text{For all } f \in Q, \text{ add } A_{pq} \to A_{pf} A_{fq}$$

Interpretation: if a string y_0 takes p with empty stack to f with empty stack and y_1 takes f with empty stack to q with empty stack then $y_0 y_1$ takes p with empty stack to q with empty stack.

`[`$x \in L(G) \Rightarrow x \in L(P)$`]`: If $A_{pq} \Rightarrow^* x$, then x can bring P from p with empty stack to a with empty stack. The proof is by induction on # steps ℓ in $A_{pq} \Rightarrow^* x$.

- Base: $\ell = 1$, then $q = p$ and $x = \epsilon$ (i.e., $A_{pp} \to \epsilon$). Obviously, ϵ can bring p with empty stack (E/S) to p with empty stack!
- Assume true for ℓ. Suppose $A_{pq} \Rightarrow^* x$ takes $\ell + 1$ steps. Two cases:

1. $A_{pq} \Rightarrow aA_{rs}b \Rightarrow^{\ell} ayb$ and $x = ayb$. By induction hypothesis, y bring r with E/S to s with empty stack. Since $A_{pq} \rightarrow aA_{rs}b$ is a rule of G, $\delta(p, a, \epsilon)$ contains $(r, \square)$ and $\delta(s, b, \square)$ contains (q, ϵ), for some stack symbol $\square$. Thus, ayb bring p with E/S to p with E/S.
2. $A_{pq} \Rightarrow A_{pf}A_{fq} \Rightarrow^{\ell} x$, then $x = y_1y_2$ and $A_{pf} \Rightarrow^{\leq \ell} y_1$ and $A_{fq} \Rightarrow^{\leq \ell} y_2$. By induction hypothesis: y_1 bring p with E/S to f with E/A, and y_2 bring f with E/S to q with E/A. Thus, y_1y_2 bring p with E/S to q with E/A.

[$x \in L(P) \Rightarrow x \in L(G)$]: Out claim is that if x can bring p with E/S to q with E/S, $A_{pq} \Rightarrow^{*} x$. The proof is by induction on # steps ℓ in going from p with E/S to q with E/S on input x.

- $\ell = 0$: then $p = q$, and $x = \epsilon$ (if $x \neq \epsilon$ or if $p \neq q$, then $\ell \geq 1$). We have $A_{pp} \rightarrow \epsilon$.
- Suppose true for ℓ. Two cases:

1. Some $\square$ is pushed in first move, and not popped until the very last move. Then, for some r, s, $(r, \square) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, \square)$ and $x = ayb$ for $a, b \in \Sigma \cup \{\epsilon\}$. Thus, we can go from r with E/A to s with E/A in ℓ steps; by induction hypothesis, $A_{rs} \Rightarrow^{*} y$. Since $A_{pq} \rightarrow aA_{rs}b$ is a production rule, $A_{pq} \Rightarrow^{*} x$.
2. $x = y_1y_2$ and stack becomes empty in a middle state f while having read y_1. By induction hypothesis, $A_{pf} \Rightarrow^{*} y_1$ and $A_{fq} \Rightarrow^{*} y_2$. Since $A_{pq} \rightarrow A_{pf}A_{fq}$, we have $A_{pq} \Rightarrow^{*} x$.

$\square$

4 First Midterm Practices

Exercise 4.1

Let $L_1 = \{ w \in \{0,1\}^* \mid w \text{ has an equal number of 0's and 1's} \}$
and $L_2 = \{ w \in \{0,1\}^* \mid w \text{ has at most 100 occurrences of 100} \}$.
Then $L_1 \cup L_2$ is context free and $\overline{L_1} \cap L_2$ is not context free. **T** **F**

Proof. False. Both L_1 and $\overline{L_1}$ are context free, and L_2 is regular. The union of two CFLs is CF. The intersection of a CFL and a regular language is CF. ☐

Exercise 4.2

Let $L_1 = \{ ww'w^R \mid w, w' \in \{0,1\}^* \}$, $L_2 = \{ ww'w \mid w, w' \in \{0,1\}^* \}$, and $L_3 = \{ w \mid w \text{ has 01 as a substring}\}$.
Then $L_1 \cap L_3$ is context free but $L_2 \cap L_3$ is not. **T** **F**

Proof. False. $L_1 = L_2 = \{0,1\}^*$. So $L_1 \cap L_3 = L_2 \cap L_3 = L_3$, and L_3 is regular! ☐

Exercise 4.3

If A and B are regular, then $C = \{ w_1 w_2 \mid w_1 \in A,\ w_2 \in B,\ |w_1| = |w_2| \}$ is necessarily context free.
T **F**

Proof. True. Think about this PDA: example 3.6. We were guessing where the middle is. ☐

Exercise 4.4

$L = \{ a^i b^j c^f d^k e^h \mid i + k = j + f + h \}$ is context free. **T** **F**

Proof. True. ☐

Exercise 4.5

Every DFA can be converted into an equivalent DFA with no incoming edges to the start state. **T** **F**

Proof. True. Make an identical copy for every state. ☐

Exercise 4.6

Give a context free grammar and a PDA for

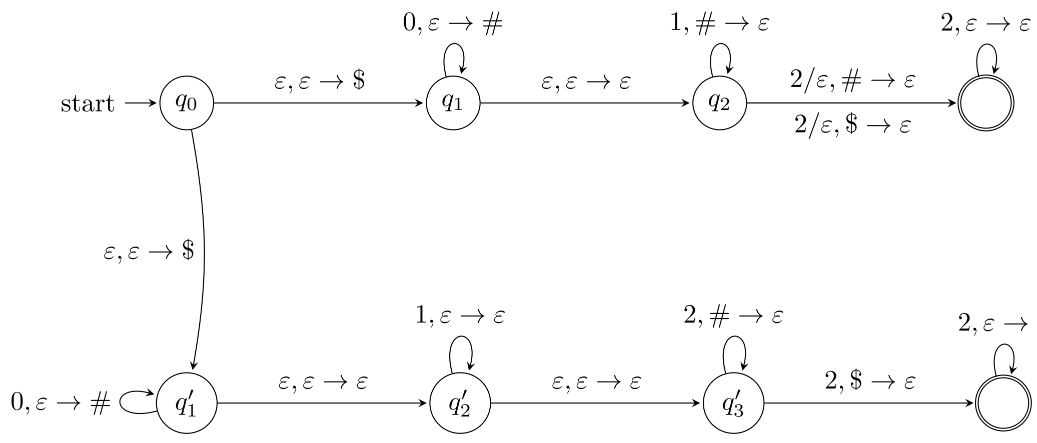
$$A = \{ 0^i 1^j 2^k \mid i \geq j \text{ or } i < k \}.$$

For example, $\epsilon \in A$ and $2 \in A$, but $1 \notin A$. No need to justify why your CFG and PDA work.

Proof. We have the following CFG

$$\begin{array}{rcl}
S & \rightarrow & V \mid U \\
V & \rightarrow & S'_0\, S'_1 \\
S'_0 & \rightarrow & 0\, S'_0\, 1 \mid 0\, S'_0 \;\mid \epsilon \\
S'_1 & \rightarrow & 2\, S'_1 \mid \epsilon \\
U & \rightarrow & U\, 2 \mid S'_2 \\
S'_2 & \rightarrow & 0\, S'_2\, 2 \mid S'_3 \\
S'_3 & \rightarrow & 1\, S'_3 \mid \epsilon
\end{array}$$

and we have the following PDA:


start
q_0
$\varepsilon, \varepsilon \rightarrow \$$
$0, \varepsilon \rightarrow \#$
q_1
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$1, \# \rightarrow \varepsilon$
q_2
$2/\varepsilon, \# \rightarrow \varepsilon$
$2/\varepsilon, \$ \rightarrow \varepsilon$
$2, \varepsilon \rightarrow \varepsilon$
$\varepsilon, \varepsilon \rightarrow \$$
$0, \varepsilon \rightarrow \#$
q'_1
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$1, \varepsilon \rightarrow \varepsilon$
q'_2
$\varepsilon, \varepsilon \rightarrow \varepsilon$
$2, \# \rightarrow \varepsilon$
q'_3
$2, \$ \rightarrow \varepsilon$
$2, \varepsilon \rightarrow \varepsilon$

□

Exercise 4.7

$\Sigma = \{a, b, c\}$ and $L = \{a^n b^n c^n : n \geq 0\}$. Is L context-free?

Proof. No, pick $a^p b^p c^p$ and prove it's not via pumping lemma. □

Exercise 4.8

What is the PDA recognizing $L = \{a^{2i} b^i : i \geq 0\}$?

Exercise 4.9

Show that if $L \subseteq \{a\}^*$ is context free, then L is regular.

5 Turing Machines: Computing with Unlimited Memory

Lecture 8 - Monday, September 29

Comment 5.1

We use $\sqcup$ to denote blank symbol. The tape initially has the input written from left to right starting from the leftmost cell, and the rest of the cells are initially filled with $\sqcup$.

Definition 5.1: Turing Machine

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where Q is the set of states, Σ: input symbols where

- Q: states, $q_0 \in Q$: start state, $q_{\text{accept}} \in Q$ accept state, $q_{\text{reject}} \in Q$ reject state.
- Σ: input symbols, where $\sqcup \notin \Sigma$.
- Γ: tape symbols, where $\sqcup \in \Gamma$. Also, $\Sigma \subseteq \Gamma$.
- $\delta\big((Q - \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma\big) \to Q \times \Gamma \times \{L, R\}$ (L = Left and R = Right)

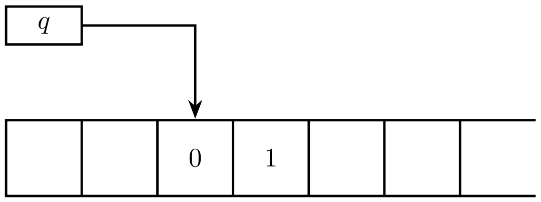
Comment 5.2

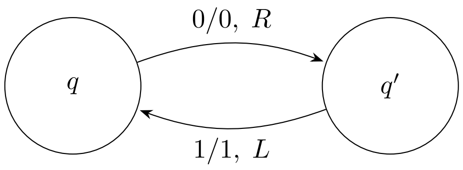
Tape head initially scanning leftmost tape square.

Possible outcomes of running $M(x)$:

- **Accept:** halting and ending up in q_{accept}.
- **Reject:** halting and ending up in q_{reject}.
- **Loops forever:** considered rejection (rejection without halting).

Example 5.1: Looping forever





Definition 5.2: Turing Recognize

We say a TM M **recognizes** language A if $L(M) = A$.

Definition 5.3: Turing Recognizable

A is **Turing recognizable** if for some TM M we have $L(M) = A$.

Definition 5.4: Decider

We say a TM M is a **decider** if it halts on all strings (i.e., never loops forever on any input).

Definition 5.5: Decide

We say a TM M **decides** language A if $L(M) = A$ and M is a decider.

Definition 5.6: Turing Decidable

A is **Turing decidable** if for some decider TM M we have $L(M) = A$.

5.1 TMs vs PDAs/ DFAs

Discovery 5.1

TMs allow us to use an infinite amount of space which we can freely access, unlike PDAs and DFAs.

Discovery 5.2

We can accept inputs under TMs without reading the input in full, but we can never do this under DFAs/ PDAs! See below for example.

Example 5.2

Suppose we have a language $L = \{x : x \text{ starts with } 1\}$. Then we can simply define the turing machine with:

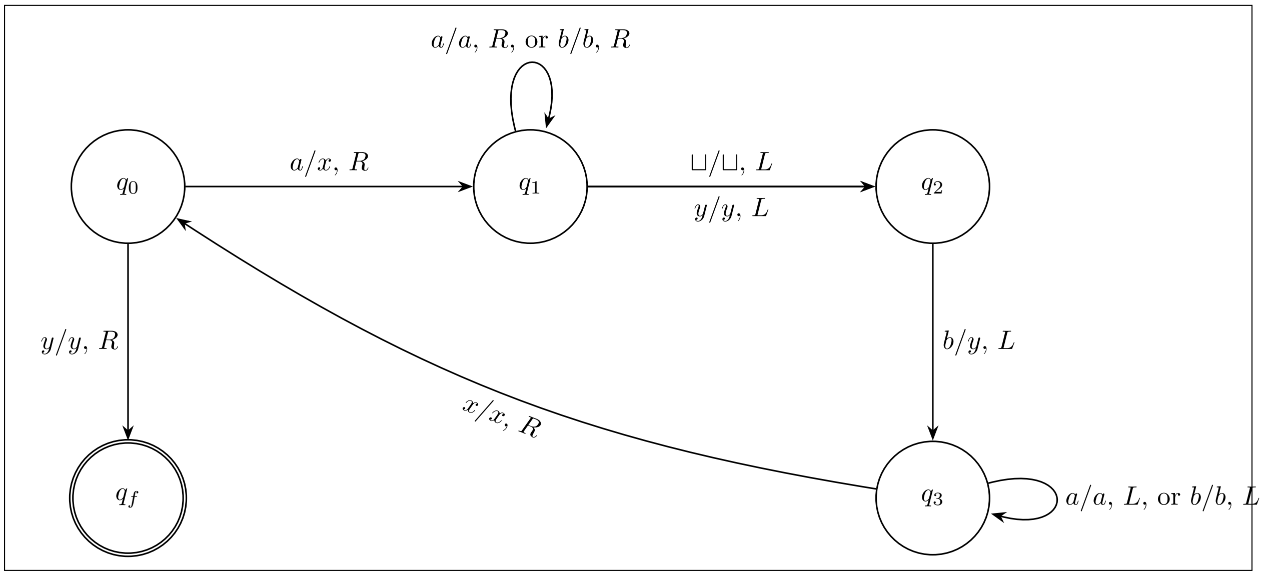
$$\delta(q_0, 0) = (q_{reject}, \#, R) \quad \text{and} \quad \delta(q_0, 1) = (q_{accept}, \#, R)$$

5.1.1 Designing Turing Machine for Languages, Examples

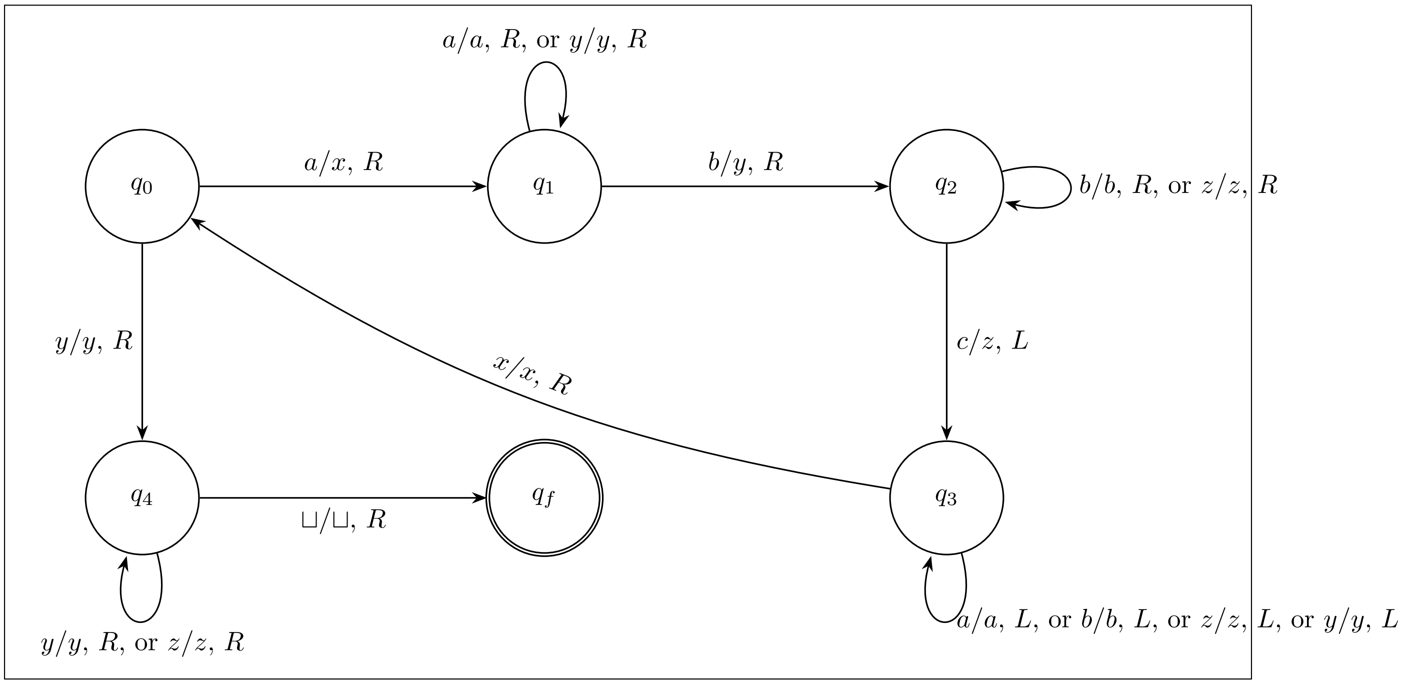
Note 5.1

If $\delta(q, a)$ for $q \notin \{q_{accept}, q_{reject}\}$ is left undefined, it moves to the reject state q_{reject}.

5.1.2 $A = \{a^i b^i : i \geq 1\}$ and its Turing Machine



5.1.3 $A = \{a^i b^i c^i : i \geq 1\}$ and its Turing Machine


a/a, R, or y/y, R
q_0
a/x, R
q_1
b/y, R
q_2
b/b, R, or z/z, R
y/y, R
c/z, L
x/x, R
q_4
$\sqcup/\sqcup$, R
q_f
q_3
y/y, R, or z/z, R
a/a, L, or b/b, L, or z/z, L, or y/y, L

5.1.4 $L = \{ww : w \in \{0,1\}^*\}$ and its Turing Machine

Exercise 5.1

This is left as an exercise. The idea is: First, mark the symbol which starts the second w, then compare each symbol of the first w with its corresponding symbol in the second w.

5.1.5 Church Turing Thesis

Theorem 5.1: Church Turing Thesis

Any function on the natural numbers can be calculated by an "effective" method if and only if it is computable by a Turing machine.

5.1.6 Variants of Turing Machines

- K tape Turing Machine
- 1-tape one-way infinite Turing Machine (Sipser's Book): A single tape for input, work, and output
- 1-tape two-way infinite TMs

Interestingly, all the above are "equivalent":

Proposition 5.1

For every K-tape TM M, there exists a 1-tape S such that $\forall\, x$, $M(x) = S(x)$.

Lecture 9 - Wednesday, October 01

Proof. S has an extended alphabet: including the alphabet Σ of M, a new symbol $\#$, and the primed version of each symbol in Σ. S contains a copy of tapes separated by $\#$, and it replaces an a in one region with an a' if the head in M points to a in that tape.

- Simulating a step in M: S locates all k marked places, remembers in its state the contents of each, and then carries out the actions according to M.
- It may need to shift whole chunks of tape contents to make room for a longer configuration.

If M takes time T on an input, S takes $O(T^2)$. □

5.2 Non-Deterministic TMs (NTMs)

Definition 5.7: Non-deterministic Turing Machine

In a **non-deterministic turing machine**, the transition is replaced by

$$\delta : Q \setminus \{q_{accept}, q_{reject}\} \times \gamma \to P(Q \times \gamma \times \{L, R\})$$

A NTM M accepts a string w if some sequence of moves from the initial ID with w as input leads to the accept state. If other sequences on w lead to nonaccept states, it's irrelevant.

Theorem 5.2

A is Turing recognizable if and only if A is accepted by an NTM TM.

5.2.1 Example $A = \{$Composite numbers$\}$

A NTM for A: On a given input n, non-deterministically choose two numbers p and q and write their binary representation on the tape. Multiply p and q on the tape, and check whether $n = pq$.

5.3 Is every language recognizable?

Question 5.1.

Is every language recognizable?

Here we show the existence of unrecognizable languages by showing that the number of TMs $\ll$ the number of languages over $\{0,1\}$. It suffices to prove that any TM may be represented as a finite string over $\{0,1\}$.

Equivalently, we can show that there exists a bijection T from the set of all natural numbers $\mathbb{N}$ to TMs (the set of all TMs or all recognizable languages is countably infinite). We also show that we cannot "list" the set of all languages (the set of all languages is uncountably infinite).

We know that there are uncountably many binary strings by the classical diagnolization technique.

Comment 5.3

We first present undecidable languages and will then show how to build unrecognizable languages.

5.3.1 First undecidable problem

For a TM M let $\langle M \rangle$ denote the encoding of M.

Theorem 5.3

SR $= \{\langle M \rangle : M$ is a TM and M doesn't accept $\langle M \rangle\}$ is undecidable (SR stands for self-reject).

Proof. Suppose SR is decidable, and N is a decider TM for SR. Since N is a decider, it halts on all given strings and either accepts or rejects and $L(N) =$ SR. What will happen if we run N on its own encoding $\langle N \rangle$?

1. If N accepts $\langle N \rangle$, then by definition of SR, we have $\langle N \rangle \notin$ SR, but we know $\langle N \rangle \in L(N)$. A contradiction.
2. If N rejects $\langle N \rangle$, then by definition of SR, we have $\langle N \rangle \in SR$, but we know $\langle N \rangle \notin L(N)$. A contradiction.

done. □

5.3.2 Acceptance problem is undecidable

Theorem 5.4

$A_{TM} = \{(\langle M \rangle, w) : M$ is a TM and M accepts $w\}$ is not decidable.

Proof. Suppose H decides A_{TM}, we can build a decider TM N for SR: $N(\langle M\rangle)$ simulatea the execution of $N(\langle M\rangle, \langle M\rangle)$ and if H accepts, reject; if H rejects, accept. Therefore, N is a decider (halts on all strings and accepts/rejects appropriately), because H is so. ☐

Question 5.2.

Is A_{TM} recognizable?

Proof. Yes, we can just run w on M. ☐

5.3.3 An Unrecognizable Language

Question 5.3.

Can we give a concrete unrecognizable language?

Proof. The language $\text{SR} = \{\langle M\rangle : M \text{ is a TM and } M \text{ doesn't accept } \langle M\rangle\}$ is in fact unrecognizable.

Suppose SR is recognizable, and N is a TM for SR. Since N is a recognizer, on $\langle M\rangle \in SR$ it halts and accepts and on other strings it either rejects, or loops forever. What will happen if we run N on its own encoding $\langle N\rangle$?

1. If N accepts $\langle N\rangle$ then by definition of SR, we have $\langle N\rangle \notin SR$, but we know that $\langle N\rangle \in L(N)$. A contradiction to $L(N) = SR$.
2. If N rejetcs $\langle N\rangle$, then by definition of SR, we have $\langle N\rangle \in SR$, but we know $\langle N\rangle \notin L(N)$. A contradiction to $L(N) = SR$.
3. If N loops on $\langle N\rangle$ then by definition of SR, we have $\langle N\rangle \in SR$, but we know $\langle N\rangle \notin L(N)$. A contradiction to $L(N) = SR$.

as desired. ☐

5.4 Relation between recognizing and deciding

Theorem 5.5

If L and $\overline{L}$ are both recognizable, then L is decidable.

Proof. Suppose M recognizes L and M' recognizes L'. We build a decider N for L: Run $M(x)$ and $M'(x)$ in parallel; if M accepts, accept; if M' accepts, reject. ☐

Theorem 5.6

If A is decidable, then $\overline{A}$ is decidable.

Theorem 5.7

If A_1 and A_2 are decidable, so are $A_1 \cap A_2$ and $A_1 \cup A_2$.

Theorem 5.8

If A_1 and A_2 are recognizable, so are $A_1 \cap A_2$ and $A_1 \cup A_2$.

5.4.1 Halting problem is not decidable

Definition 5.8: Halting Problem

The **Halting problem** is $A_{halt} = \{(\langle M \rangle, w) : \langle M \rangle \text{ is a TM and } M \text{ halts on } w\}$.

The halting problem is clearly recognizable, since we can just run $\langle M \rangle$ with input w. However, it is not decidable:

Theorem 5.9

Halting problem is not decidable.

Proof. Suppose, for contradiction, that the halting problem is decidable. Then there is a TM

$$H(\langle M \rangle, w) = \begin{cases} \text{accept} & \text{if } M \text{ halts on input } w, \\ \text{reject} & \text{if } M \text{ does not halt on input } w. \end{cases}$$

Using H, define a new TM D which, on input $\langle M \rangle$, does the following:

$$D(\langle M \rangle): \quad \text{Run } H(\langle M \rangle, \langle M \rangle). \begin{cases} \text{If } H \text{ accepts, then } \textit{loop forever.} \\ \text{If } H \text{ rejects, then } \textit{accept.} \end{cases}$$

Now analyze D on its own description $\langle D \rangle$. (1) If $H(\langle D \rangle, \langle D \rangle)$ accepts, then by definition D halts on $\langle D \rangle$. But then D's code says to *loop forever* in this case, a contradiction.

(2) If $H(\langle D \rangle, \langle D \rangle)$ rejects, then by definition D does *not* halt on $\langle D \rangle$. But then D's code says to *accept* in this case, so D halts, again a contradiction. □

5.4.2 An undecidable language not involving TMs

Theorem 5.10

$ALL_{cfg} = \{G : G \text{ is a CFG and } L(G) = \Sigma^*\}$ is undecidable.

Proof. Let A^*_{TM} be $\{(\langle M \rangle, w) : \langle M \rangle \text{ is a TM and } M \text{ does not accept } w\}$, and let $S = \{(\langle M \rangle, w) : M \text{ is a TM}\}$. Hence we know that

$$A_{TM} = \overline{A^*_{TM}} \cap S$$

If A^*_{TM} is decidable, then $\overline{A^*_{TM}}$ is decidable. Because S is also decidable, then A_{TM} is also decidable. However, recall that acceptance problem is undecidable (see section 5.3.2), so we know that A^*_{TM} has to be undecidable.

We wish to design a PDA P such that $L(P) = \Sigma^*$ if and only if $(M, w) \in A^*_{TM}$. We reduce from the undecidable language

$$A^*_{TM} = \{\, \langle M, w\rangle : M \text{ does } \textit{not} \text{ accept } w\,\}$$

to ALL_{cfg}. Given $\langle M, w\rangle$, we effectively build a PDA (equivalently a CFG) P such that

$$L(P) = \Sigma'^* \quad\Longleftrightarrow\quad \langle M, w\rangle \in A^*_{TM},$$

where Σ' is the alphabet we define below. This gives a mapping reduction $A^*_{TM} \leq_m ALL_{cfg}$ and hence proves ALL_{cfg} undecidable.

Computation histories. Let Γ be the tape alphabet of M and Q its set of states. A *configuration* of M is a string over $\Gamma \cup Q$ with exactly one state symbol indicating the head position. Fix a separator symbol # not in $\Gamma \cup Q$ and put $\Sigma' = \Gamma \cup Q \cup \{\#\}$. A string

$$H = C_0\#C_1\#\cdots\#C_k \in (\Sigma')^*$$

is an *accepting computation history of* M *on* w if: (i) C_0 is the initial configuration on input w, (ii) C_k is an accepting configuration, and (iii) for each i, $C_i \vdash C_{i+1}$ is a single valid move of M.

Let $\mathrm{ACC}(M, w)$ denote the set of all such histories (possibly empty).

A CFL covering all "bad" strings. We construct a CFL B (via a PDA) such that

$$B = (\Sigma')^* \quad \text{iff} \quad \mathrm{ACC}(M, w) = \varnothing,$$

and otherwise $B = (\Sigma')^* \setminus \mathrm{ACC}(M, w)$. We obtain B as a finite union of CFLs capturing different ways a string fails to be a valid accepting history:

$$\begin{aligned}
B_1 &= \{\text{strings not of the block form } C_0\#C_1\#\cdots\#C_k\},\\
B_2 &= \{\text{strings whose first block } C_0 \text{ is not the correct initial config on } w\},\\
B_3 &= \{\text{strings whose last block } C_k \text{ is not an accepting configuration}\},\\
B_4 &= \{\text{strings where for some } i, \text{ the step } C_i \not\vdash C_{i+1}\}.
\end{aligned}$$

The languages B_1, B_2, B_3 are regular (hence context-free). For B_4, we use the standard *local-check* construction: a PDA can nondeterministically choose an index i and a head neighborhood and verify, symbol-by-symbol, that C_{i+1} does *not* follow from C_i by one move of M (split into finitely many subcases for left/right moves and tape updates). Each subcase is context-free; their union is context-free. Thus B_4 is CFL. Let

$$B \;=\; B_1 \,\cup\, B_2 \,\cup\, B_3 \,\cup\, B_4,$$

which is a finite union of CFLs and hence a CFL. We can effectively construct from $\langle M, w\rangle$ a PDA P with $L(P) = B$.

Correctness of the construction. If M does *not* accept w, then there is no accepting history, i.e., $\mathrm{ACC}(M, w) = \varnothing$. Every string necessarily violates at least one of the conditions (i)–(iii), hence lies in B, so $L(P) = B = (\Sigma')^*$.

If M *does* accept w, let $H \in \mathrm{ACC}(M, w)$ be a valid accepting history. Then $H \notin B$ (it violates none of $B_1, \ldots, B_4$), so $L(P) = B \subsetneq (\Sigma')^*$. Therefore

$$L(P) = \Sigma'^* \quad\Longleftrightarrow\quad M \text{ does not accept } w.$$

Conclusion. From $\langle M, w\rangle$ we computably obtain a CFG G (equivalent to P) such that $L(G) = \Sigma'^*$ iff $\langle M, w\rangle \in A^*_{TM}$. If ALL_{cfg} were decidable, this would decide A^*_{TM}, contradicting the undecidability of A^*_{TM}. Hence ALL_{cfg} is undecidable. $\square$

Lecture 11 - Wednesday, October 08

In class midterm today.

Lecture 12 - Monday, October 20

Now the question becomes:

Question 5.4.

Is ALL_{cfg} recognizable?

We know that the set $\overline{ALL_{cfg}}$ is recognizable because there exists a polynomial time algorithm D such that

$$D(G, x) = 1 \quad \text{if and only if} \quad x \in L(G)$$

but because ALL_{cfg} is undecidable, so ALL_{cfg} cannot be recognizable.

5.4.3 More Undecidable Problems

Definition 5.9: Post Correspondence Problem

The problem takes input a set of tiles of the form $\left[\frac{u}{v}\right]$, where u and v are strings over Σ^*. The question is, can we find a sequence $\left[\frac{u_1}{v_1}\right], \ldots, \left[\frac{u_k}{v_k}\right]$ such that $u_1 \cdots u_k = v_1 \cdots v_k$.

Theorem 5.11

The Post Correspondence Problem is undecidable.

5.5 Doubly Unrecognizable Problem

There are two proofs:

- Non-constructive proof: the number of recognizable languages and co-recognizable languages is countable, while the number of languages in total is uncountable, so there must exists languages that are doubly unrecognizable.
- Constructive proof.

Example 5.3

Consider the language A defined as

$$A = \{(\langle M_1, M_2 \rangle) : M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

We claim that A is doubly unrecognizable.

[A **is unrecognizable**] Given a recognizer R for A, we build a recognizer R' for $\overline{A_{TM}}$. Let M_{empty} be a TM for the empty language. $R'(M, w)$ does the following:

1. Let M_w be a TM that on any input x, erases x and stimulates $M(w)$;
2. Simulate $R(M_w, M_{empty})$.

Then

$$(M, w) \in \overline{A_{TM}} \iff M \text{ does not accept } w \iff L(M_w) = \emptyset \iff (M_W, M_{empty}) \in A$$

[$\overline{A}$ **is unrecognizable**] Define language B as

$$B = \{(\langle M_1, M_2 \rangle) : M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) \neq L(M_2)\}$$

so B is recognizable if and only if $\overline{A}$ is recognizable, so it suffices for us to show that B is unrecognizable. Given a recognizer R for B, we build a recognizer R' for $\overline{A_{TM}}$. Let M_{all} be a TM for the language of all strings. $R'(M, w)$ does the following:

- Let M_w be a TM that on any input x, erases x and stimulates $M(w)$;
- Stimulate $R(M_w, M_{all})$.

Then

$$(M, w) \in \overline{A_{TM}} \iff M \text{ does not accept } w \iff L(M_w) = \emptyset \iff (M_W, M_{all}) \in B$$

Example 5.4

Consider the language

$$L = \{(0, G_1, G_2) : L(G_1) = L(G_2)\} \cup \{(1, G_1, G_2) : L(G_1) \neq L(G_2)\}$$

It is easy to see that L is doubly unrecognizable.

6 Computation Complexity

Definition 6.1: Complexity Theory

Complexity theory categorizes decidable problems in terms of how difficult it is to solve them.

Definition 6.2: (Worst Case) Running Time

A deterministic TM M has a **(worst case) running time** (or time complexity) $t(n)$ if whenever M is given an input w of length n, M halts after making at most $t(n)$ moves, regardless of whether M accepts.

Comment 6.1

Different models of TMs give rise to different running times for a problem, but they are polynomially related. (By default, we measure in terms of single-tape TM).

Example 6.1

$A = \{0^n 1^n\}$ can be decided in time $O(n^2)$ (or $O(n \log n)$ with more efforts) on a single-tape TM, but not in time $O(n)$ on single-tape TMs.

Note 6.1

A $t(n)$ time multitape TM can be simulated via a $O(t(n)^2)$ time single-tape TM.

Lecture 13 - Wednesday, October 22

Definition 6.3: Complexity Class

The **complexity class** $DTIME(T(n))$ is the class of languages decided by a single-tape deterministic TM which always halts in $O(T(n))$ time.

Note 6.2

P is the class of languages that are decidable in polynomial time on a deterministic single-tape TM. i.e.,

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} DTIME(n^k)$$

6.1 What is potytime and what can be solved in polytime?

Definition 6.4: Polytime

A TM is **poly-time** if it's running time is $O(n^k)$ for some k.

Thesis: Deterministic poly-time TM's and the class P adequately capture the intuitive notions of practically feasible algorithms, and realistically solvable problems, respectively.

6.1.1 Examples of Languages in P

Example 6.2

"Given a graph: is it connected?" – `This belongs to P`.

Example 6.3

"Given a number: is it prime?" – `This belongs to P due to a very smart AKS algorithm`.

Example 6.4

$CFL \subseteq \mathsf{P}$. If L is a CFL, it has a CFG G; we can test membership $x \in^? L(G)$ in $O(|x|^3)$.

6.1.2 Time Hierarchy Theorem

Theorem 6.1: Time Hierarchy Theorem

Let **Dtime**$(T(\cdot))$ be the set of languages that are decidable in time $O(T(n))$ by a Turing Machine. Then for all $k \in \mathbb{N}$:

$$\mathbf{Dtime}(n^k) \subsetneq \mathbf{Dtime}(n^{k+1})$$

Comment 6.2

Informally, the theorem states that in any "reasonable" model of computation (Turing Machine, RAM, etc.) where computation could be emulated, having sufficiently more "time" implies more "computational power".

To prove the above theorem, we will need to introduce universal Turing Machine first.

Theorem 6.2: Efficient Universal Turing Machine

There exists a single-tape TM $\mathcal{U}$ such that for every $x, \alpha \in \{0,1\}^*, \mathcal{U}(x,\alpha) = M_\alpha(x)$, where M_α denotes the TM represented by α.

Moreover, if M_α halts on input x within T steps, then $\mathcal{U}(x,\alpha)$ halts within $CT\log T$ steps, where C is a number independent of $|x|$ and depending only on M_α's alphabet size, number of tapes, and number of states.

Proof of Time Hierarchy Theorem 6.1. The proof is by diagonalization. Fix $k \in \mathbb{N}$. Let

$$A = \left\{ \langle M \rangle : M(\langle M \rangle) \text{ rejects in } n^{k+(1/2)} \text{ steps where } n = |\langle M \rangle| \right\}$$

[$A \in \mathbf{Dtime}(n^{k+1})$] Given an input $\langle M \rangle$, simulate M on $\langle M \rangle$ for $n^{k+(1/2)}$ steps and flip the reject/ accept (if it does not halt, we simply reject). This can be done in time $O(n^{k+(1/2)} \log n)$, and hence $O(n^{k+1})$ on a deterministic universal TM.
[$A \notin \mathbf{Dtime}(n^k)$] Suppose to the contrary N decides A in time $O(n^k)$. On any $x \in A$, $N(x)$ accepts (halts in q_{accept} in $O(n^k)$ time), and on any input $x \notin A$, $N(x)$ rejects (halts in q_{reject} in $O(n^k)$ time). We have two cases

1. $N(\langle N \langle)$ rejects in time $O(n^k)$, then $\langle N \rangle \in A$, contradicting $L(N) = A$;
2. $N(\langle N \langle)$ accepts in time $O(n^k)$, then $\langle N \rangle \notin A$, contradicting $L(N) = A$;

□

6.2 Is it easier to verify a solution than to find it?

6.2.1 Class NP (Non-Deterministic Polynomial Time)

Definition 6.5: NP

A language $L \subseteq \{0,1\}^*$ is in **NP** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a poly-time DTM M (called a **verifier**) such that for all $x \in \{0,1\}^*$:

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1, \text{ where we call } u \text{ a } \textbf{witness} \text{ or } \textbf{ciertificate}.$$

Example 6.5

The language of graph isomorphism, $L = \{(G_1, G_2) : G_1 \cong G_2\}$, is in NP.

- input x represents the adjacency matrices of G_1 and G_2;
- input u represents a potential mapping of the nodes in G_1 and G_2;
- $M(x,u)$ checks if the mapping is an isomorphism.

Example 6.6

The language of graph hamiltonicity, $L = \{G : G \text{ has a Hamiltonian path}\}$, is in NP.

- input x represents the adjacency matrices of G_1 and G_2;
- input u represents a potential Hamiltonian path in G;
- $M(x,u)$ checks if u is a path and visits every node exactly once;
- Note that $|u| \leq poly(|x|)$.

Example 6.7

Satisfiability (SAT) problem: Consider the language $L = \{\phi : \phi \text{ is satisfiable}\}$, where x represents a Boolean formula and u a Boolean assignment, and M checks if the assignment satisfie the fomula ($|u| \in poly(|x|)$ and M works in polytime). This is in NP.

Question 6.1.

Is Unsatisfiability in NP? $\mathsf{UNSAT} = \{\phi : \phi \text{ is not satisfiable}\}$.

Answer. This is not known yet. □

Lecture 14 - Monday, October 27

6.2.2 P = NP?

Conjecture: $\mathsf{P} \neq \mathsf{NP}$.

6.3 Reductions: Comparing Relative Hardness

Definition 6.6: Karp Reduction

L is **Karp-reduced** to L' (denoted $L \leq_p L'$) if there exists a poly-time computable f such that

$$\forall x : x \in L \iff f(x) \in L'$$

Example 6.8: CLIQUE $\leq_p$ V-COVER

A clique is a set of nodes that are all connected. `CLIQUE` asks if a given graph $G = (V, E)$ has a clique of size k, and `V-COVER` asks if a given graph $G = (V, E)$ have a vertex cover of size s.

We have the following lemma:

Lemma 6.1

Given $G = (V, E)$ and let $\overline{G}$ be the complement, then

$$(G, k) \in \texttt{CLIQUE} \iff (\overline{G}, n - k) \in \texttt{V-COVER}$$

Hence we can define our f to be $f(G, k) = (\overline{G}, n - k)$ for all (G, k). It is easy to verify that f is poly-time computable.

6.3.1 NP hardness and NP completeness

Definition 6.7: NP-hard

We say that L' is **NP-hard** if $L \leq_p L'$ for every $L \in \mathsf{NP}$.

Definition 6.8: NP-complete

We say that L' is **NP-complete** if L' is **NP-hard** and $L' \in \mathsf{NP}$.

Theorem 6.3: Cook 1971, Levin 1973

$\texttt{SAT} = \{\phi : \phi \text{ is in CNF and } \phi \text{ is satisfiable}\}$ is NP-complete.

Theorem 6.4: Transitivity

If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$, then $L_1 \leq L_3$.

Proof. There exists poly-time functions f, g such that $x \in L_1$ if and only if $f(x) \in L_2$ and that $x' \in L_2$ if and only if $g(x') \in L_3$. Thus, $x \in L_1$ if and only if $g(f(x)) \in L_3$. Also $g(f(\cdot))$ is poly-time computable because both f and g are. □

Result 6.1

As a result, if $\texttt{SAT} \leq_p L$, then L is NP-hard. Moreover, if L is in NP, then L is NP-complete.

Theorem 6.5

If $L' \leq_p L$ and $L \in \mathsf{P}$, then $L' \in \mathsf{P}$.

Theorem 6.6: Collapsing Theorem

If L is NP-complete and $L \in \mathsf{P}$, then $\mathsf{P} = \mathsf{NP}$.

6.3.2 Proving NP-completeness

Define $k\texttt{SAT}$ to be the language whose clauses has exactly k literals. In other words, for $\varphi \in k\texttt{SAT}$, then $\varphi = C_1 \wedge \cdots \wedge C_m$, where each C_i is of the form $C_i = \ell_1 \vee \cdots \vee \ell_k$ for literals ℓ_j. We have

- $\texttt{1SAT} \in P$; and
- $\texttt{2SAT} \in P$; but
- $\texttt{3SAT}$ is NP-complete.

Lecture 15 - Wednesday, October 29

$\texttt{SAT} \leq_p \texttt{3SAT}$ Our goal is to show that, given a CNF formula $\varphi = C_1 \wedge \cdots \wedge C_m$ (clauses are disjunctions of literals), produce in polynomial time an equi-satisfiable CNF φ' in which *every* clause has exactly three literals.

Comment 6.3

Assume fresh variables introduced below do not appear elsewhere.

We have three cases:

1 `[Two-literal clause.]` If $C = (x_1 \vee x_2)$, introduce a fresh variable p and replace C by

$$(x_1 \vee x_2 \vee p) \;\wedge\; (x_1 \vee x_2 \vee \neg p)$$

(Always satisfied exactly when $x_1 \vee x_2$ is satisfied.)

2 `[Unit clause.]` If $C = (x)$, introduce fresh p_1, p_2 and replace C by the four clauses

$$(x \vee p_1 \vee p_2) \;\wedge\; (x \vee p_1 \vee \neg p_2) \;\wedge\; (x \vee \neg p_1 \vee p_2) \;\wedge\; (x \vee \neg p_1 \vee \neg p_2)$$

(These force satisfaction iff x is true.)

3 `[Clause with` $r > 3$ `literals.]` If $C = (z_1 \vee z_2 \vee \cdots \vee z_r)$, introduce fresh variables $y_1, \ldots, y_{r-3}$ and replace C by the chain

$$(z_1 \vee z_2 \vee y_1) \;\wedge\; (\neg y_1 \vee z_3 \vee y_2) \;\wedge\; (\neg y_2 \vee z_4 \vee y_3) \;\wedge\; \cdots \;\wedge\; (\neg y_{r-4} \vee z_{r-2} \vee y_{r-3}) \;\wedge\; (\neg y_{r-3} \vee z_{r-1} \vee z_r).$$

(This set of $r-2$ three-literal clauses is satisfied iff the original long clause is satisfied.)

Exercise 6.1

Prove correctness of each replacement and that the construction is polynomial.

`2SAT` $\in P$ The main idea is that

$$(x \vee y) \wedge (\overline{x} \vee y) \equiv (y \vee z) \qquad \text{if } x \notin \{y, z, \overline{y}, \overline{z}\}$$

`3SAT` $\leq_p$ `CLIQUE` Let

$$\Phi \;=\; (x_{1,1} \vee x_{1,2} \vee x_{1,3}) \;\wedge\; (x_{2,1} \vee x_{2,2} \vee x_{2,3}) \;\wedge\; \cdots \;\wedge\; (x_{k,1} \vee x_{k,2} \vee x_{k,3})$$

be a 3-CNF with k clauses, where each $x_{i,j}$ is a literal (a variable or its negation). We construct a graph G such that

$$\Phi \in \texttt{3SAT} \quad\Longleftrightarrow\quad (G, k) \in \texttt{CLIQUE}$$

We construct G in the following way:

- **Nodes.** For each literal $x_{i,j}$ (the j-th literal in clause i), create a node labeled $x_{i,j}$. There are $3k$ nodes total.
- **Edges.** For two nodes $x_{i,j}$ and $x_{i',j'}$, add the edge $\{x_{i,j}, x_{i',j'}\}$ *iff*

$$i \neq i' \quad \text{and} \quad x_{i,j} \neq \overline{x_{i',j'}}.$$

(So: connect literals from different clauses unless they are complementary.)

Note 6.3

The construction makes $3k$ nodes and $\mathcal{O}(k^2)$ edge checks, hence is polynomial.

Result 6.2

Since we showed $\texttt{SAT} \le_p \texttt{3SAT} \le_p \texttt{CLIQUE} \le_p \texttt{V-Cover}$. All these problems are in NP. And since $\texttt{SAT}$ is NP-complete, all of them are NP complete.

6.3.3 Cook-Levin Theorem

Theorem 6.7: Cook-Levin Theorem

For every language $L \in \mathsf{NP}$ there exists a polynomial-time computable map f such that for all inputs x, $x \in L$ iff $f(x)$ is a satisfiable CNF formula. Equivalently, $\texttt{SAT}$ is NP-complete.

We have seen that $\texttt{SAT} \in$ NP. We now show that for any NP language L we have $L \le_p \texttt{SAT}$.

Cook–Levin: $\mathsf{NP} \le_p$ SAT. Fix any language $L \in \mathsf{NP}$. There exists a single-tape nondeterministic Turing machine $M = (Q, \Gamma, \Sigma, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$ and a polynomial $T(n)$ such that on every input $x \in \Sigma^n$, M halts within $T(n)$ steps and $x \in L$ iff M accepts x. We build, in time poly(n), a CNF formula Φ_x that is satisfiable iff M accepts x within $T(n)$ steps.

Tableau and variables. Let the computation tableau have rows $t = 0, 1, \dots, T$ (time) and columns $i = 1, 2, \dots, S$ (tape cells), where $T := T(n)$ and $S := T + 1$ suffice for a single-tape machine. Introduce Boolean variables

$$X_{t,i,\sigma} \quad (\sigma \in \Gamma), \qquad H_{t,i}, \qquad Q_{t,q} \quad (q \in Q),$$

intended to mean: at time t, cell i contains σ, the head is at cell i, and the control state is q, respectively.

CNF constraints (conjoined to form Φ_x). *(A) Exactly-one constraints (well-formed configurations).* For each t and i:

$$\bigvee_{\sigma \in \Gamma} X_{t,i,\sigma} \;\wedge\; \bigwedge_{\sigma \ne \sigma'} (\neg X_{t,i,\sigma} \vee \neg X_{t,i,\sigma'}),$$
$$\bigvee_{i=1}^{S} H_{t,i} \;\wedge\; \bigwedge_{i \ne j} (\neg H_{t,i} \vee \neg H_{t,j}),$$
$$\bigvee_{q \in Q} Q_{t,q} \;\wedge\; \bigwedge_{q \ne q'} (\neg Q_{t,q} \vee \neg Q_{t,q'}).$$

(B) Initial configuration. At $t = 0$ the state is q_0, the head is at cell 1, the tape spells x then blanks:

$$Q_{0,q_0} \;\wedge\; H_{0,1} \;\wedge\; \bigwedge_{i=1}^{|x|} X_{0,i,x_i} \;\wedge\; \bigwedge_{i=|x|+1}^{S} X_{0,i,\texttt{blank}}.$$

(C) Acceptance within T steps.

$$\bigvee_{t=0}^{T} Q_{t,q_{\mathrm{acc}}}.$$

(D) Local transition (consistency) constraints. For each $t \in \{0, \dots, T-1\}$ and each i:

(D1) Non-head cells keep their symbol. For every $\sigma \in \Gamma$,

$$(H_{t,i} \vee \neg X_{t,i,\sigma} \vee X_{t+1,i,\sigma}) \ \wedge \ (H_{t,i} \vee \neg X_{t+1,i,\sigma} \vee X_{t,i,\sigma}).$$

(D2) Head cell updates according to δ. For each transition $\delta(q,\sigma) \ni (q', \tau, d)$ with $d \in \{-1, 0, +1\}$ and all i with $1 \le i, i+d \le S$:

$$\begin{aligned}&(\neg Q_{t,q} \vee \neg H_{t,i} \vee \neg X_{t,i,\sigma} \vee Q_{t+1,q'}),\\ &(\neg Q_{t,q} \vee \neg H_{t,i} \vee \neg X_{t,i,\sigma} \vee X_{t+1,i,\tau}),\\ &(\neg Q_{t,q} \vee \neg H_{t,i} \vee \neg X_{t,i,\sigma} \vee H_{t+1,i+d}).\end{aligned}$$

To forbid illegal moves, for any triple (q, σ) and displacement d not allowed by δ, add

$$\neg Q_{t,q} \vee \neg H_{t,i} \vee \neg X_{t,i,\sigma} \vee \neg H_{t+1,i+d}.$$

Size bound and construction time. We have $(T+1)S = \mathcal{O}(T^2)$ time–cell pairs, and the numbers of symbols $|\Gamma|$ and states $|Q|$ are constants for M. Hence the variable and clause counts are poly(n), and generating all clauses is doable in poly(n) time.

Correctness. $(\Rightarrow)$ If M accepts x in $\le T$ steps, interpret each variable by the accepting tableau: $X_{t,i,\sigma}$ is true exactly for the symbol present, and similarly for $H_{t,i}$ and $Q_{t,q}$. Then (A) holds by construction, (B) holds at $t = 0$, (D1)–(D2) hold because successive rows follow δ, and (C) holds since an accepting row exists; thus Φ_x is satisfiable.

$(\Leftarrow)$ If Φ_x is satisfiable, (A) forces each row to encode exactly one configuration (unique symbol per cell, unique head position, unique state). Clauses (B) and the $(t \to t{+}1)$ clauses in (D1)–(D2) ensure that successive rows follow the transition function δ from the correct start row. Clause (C) guarantees that some row is in state q_{acc}. Therefore M accepts x within T steps.

Conversion to 3-CNF. Every clause above has constant width; convert to 3-CNF via the standard introduce-a-fresh-variable gadget, incurring only a linear blow-up. Hence the mapping $x \mapsto \Phi_x$ is a polynomial-time reduction from L to SAT.

This proves that SAT is NP-hard. Since $\mathsf{SAT} \in \mathsf{NP}$, SAT is NP-complete. $\square$

6.3.4 Alternative definition of NP and Non-deterministic Turing Machines

We may define NP in terms of poly-time non-deterministic TMs.

Comment 6.4

We will work with TMs with a branching factor of two. The NDTM has two transition functions $\delta_0 : Q \times \Gamma \to \mathbb{Q} \times \Gamma \times \{L, R\}$ and $\delta_1 : \mathbb{Q} \times \Gamma \to \mathbb{Q} \times \Gamma \times \{L, R\}$.

Definition 6.9: Runtime of NDTM

Let N be a nondeterministic TM all of whose branches halt. The running time of N, denoted $t(n)$, is the maximum number of steps that N uses on any branche of its computation on any input of length n.

Definition 6.10: NTIME$(T(n))$

NTIME$(T(n))$ is the class of languages decided by a single tape non-deterministic $O(T(n))$ TM.

Note 6.4

A $t(n)$ time nondeterministic TM can be simulated by a $O(2^{O(t(n))})$ time deterministic TM on three tapes (and hence the same time complexity on a single tape).

Theorem 6.8

We have

$$\texttt{NP} = \bigcup_{c\in\mathbb{N}} \texttt{NTIME}(n^c)$$

Proof. [$\Longleftarrow$] Given a poly-time DTM verifier V and poly p, define NTM N: on input x, nondeterministically guess $u \in \{0,1\}^{p(|x|)}$ and accept iff $V(x,u) = 1$. Then N runs in poly-time and $x \in L \Leftrightarrow N$ accepts x, so $L \in \mathsf{NP}$.

[$\Longrightarrow$] Given $L \in \mathsf{NP}$, let NTM M decide L in time $T(n) \leq n^c$ with binary branching. Define DTM V: on input (x,u) with $|u| = T(|x|)$, simulate $M(x)$ for $\leq T(|x|)$ steps, fixing each nondeterministic choice by the next bit of u; accept iff the run reaches q_{acc}. Then V runs in poly-time and

$$x \in L \iff \exists u \in \{0,1\}^{T(|x|)}\ V(x,u) = 1.$$

Thus L has a polynomial-time verifier. □

Lecture 16 - Monday, November 03

Proposition 6.1

If $L_1 \in \text{NP}$ and $L_0 \leq_p L_1$, then $L_0 \in \text{NP}$.

Proof. Since $L_1 \in NP$, $\exists$ DTM verifier M_1: $x \in L_1 \iff \exists u\ M_1(x,u) \in L_1$. Since $L_0 \leq_p L_1$, $\exists$ poly-time computable p: $x \in L_0 \iff p(x) \in L_1$. DTM verifier M_0 for L_0: $M_0(x,u)$ simulate $M_1(p(x),u)$. □

6.3.5 Class coNP

Definition 6.11: coNP

A language L is in **coNP** if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a poly-time TM M such that for all $x \in \Sigma(L)^*$,

$$x \in L \iff \forall u \in \Sigma(L)^{\leq p(|x|)},\ M(x,u) = 1$$

Proposition 6.2

We have $L \in \text{coNP}$ if and only if $\overline{L} \in \text{NP}$.

Theorem 6.9

If L is NP-complete, then $\overline{L}$ is coNP-complete.

Proof. First, since $L \in \text{NP}$, by definition of coNP we have $\overline{L} \in \text{coNP}$.

It remains to show coNP-hardness of $\overline{L}$. Notice that for all $L' \in \text{coNP}$, we have

$$x \in L' \iff x \notin \overline{L'} \iff p(x) \notin L \iff p(x) \in \overline{L}$$

□

Example 6.9

Here are some examples in coNP: non-satisfiability, tautologies, non-hamiltonicity

Definition 6.12: coNP-complete

We say L is **coNP-complete** if $L \in \text{coNP}$ and for all $A \in \text{coNP}$, we have $A \leq_p L$.

Theorem 6.10

Non-satisfiability is coNP-complete.

Lecture 17 - Wednesday, November 05

Theorem 6.11

Unless NP = coNP, `SAT` $\notin$ coNP.

Proof. We argue by contradiction. Assume `SAT` $\in$ coNP.

Step 1: NP $\subseteq$ coNP. Since `SAT` is NP-complete under Karp (many-one) reductions, for every $L \in$ NP there exists a polynomial-time computable f such that

$$x \in L \iff f(x) \in \texttt{SAT}.$$

Because `SAT` $\in$ coNP and coNP is closed under Karp preimages (i.e., if $A \in$ coNP and $B \le_m^p A$, then $B \in$ coNP), it follows that $L \in$ coNP. Hence NP $\subseteq$ coNP.

Step 2: coNP $\subseteq$ NP. From `SAT` $\in$ coNP we get $\overline{\texttt{SAT}} = \texttt{UNSAT} \in$ NP. We claim `UNSAT` is coNP-complete under Karp reductions: for any $A \in$ coNP, the complement $\overline{A} \in$ NP, so there is a polynomial-time g with

$$x \in \overline{A} \iff g(x) \in \texttt{SAT}.$$

Taking complements gives

$$x \in A \iff g(x) \in \overline{\texttt{SAT}} = \texttt{UNSAT},$$

so $A \le_m^p$ `UNSAT`. Since `UNSAT` $\in$ NP and NP is closed under Karp preimages, we conclude $A \in$ NP. Thus coNP $\subseteq$ NP.

Combining the two inclusions yields NP = coNP, contradicting the premise. Therefore, unless NP = coNP, we must have `SAT` $\notin$ coNP. □

Corollary 6.1

If P = NP $\cap$ coNP, then we can solve the `Factoring` problem in polynomial time.

Proof. Consider the decision language

$$L_{\le} = \{(n,k) \in \mathbb{N}^2 : \exists \text{ prime } r \le k \text{ with } r \mid n\}.$$

We claim $L_{\le} \in \mathbf{NP} \cap \mathbf{coNP}$.

NP: A YES witness is such a prime $r \le k$ with $r \mid n$. Verification is polynomial time: check $r \le k$, test r is prime (AKS), and test $r \mid n$.

coNP: A NO witness certifies that n has no prime factor $\le k$. Since the input is promised semiprime $n = pq$, a succinct certificate is: the prime factorization $n = p^1q^1$ together with $p > k$ and $q > k$ is *not* possible if $k \ge \sqrt{n}$ (because then $\min\{p,q\} \le \sqrt{n} \le k$). Instead, for general k, give the factorization $n = pq$ and show $p > k$ and $q > k$ fail—but this cannot hold. A standard workaround is to use Pratt/AKS certificates to show primality of *all* primes $\le k$ and then certify that none divides n. Concretely, list all primes $\le k$ together with succinct primality certificates and give the remainders $n \bmod r$ for each such r, each nonzero. The number of primes $\le k$ is $O(k/\log k)$ and each check is polynomial in $\log n$ and $\log k$, so the aggregate certificate is polynomial in the input size when $k \le n$. Thus $L_{\le} \in \mathbf{coNP}$.

Hence, under $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$, we have $L_{\le} \in \mathbf{P}$.

Now solve semiprime factoring in polynomial time using $L_{\leq}$ as a subroutine. Given $n = pq$ with $p \leq q$, note $p \leq \sqrt{n}$. Perform binary search on $k \in [2, \lfloor\sqrt{n}\rfloor]$ using the predicate $(n, k) \in L_{\leq}$, which is monotone in k. In $O(\log n)$ queries, we find the least prime factor p of n. Set $q \leftarrow n/p$ and output (p, q). All arithmetic is on $O(\log n)$-bit integers, and each call to $L_{\leq}$ runs in polynomial time, so the overall runtime is polynomial in $\log n$.

Therefore, assuming $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$, semiprime factoring (outputting the two primes) is in polynomial time. □

6.3.6 Search vs. Decision for SAT

So far we defined problems as "decision problems" Now we care about: given formula Φ, we wish to find an assignment A such that $\Phi(A) = 1$. In other words, it does not just tell us whether or not A exists or not, we are looking for one.

Let `Search-SAT` be search version of `SAT`.

It is easy to see that if we can solve `Search-SAT` in polynomial time, then we can solve (Decision) `SAT` in polynomial time as well.

Comment 6.5

The reduction is very similar to Karp-Reduction, but note that `Search-SAT` is **not** a decision problem.

Question 6.2. .

What if we can solve `Decision-SAT` $\in$ P? Can we then solve `Search-SAT` as well?

Theorem 6.12

If we could solve (Decision) `SAT` in polynomial time we can also solve the search version, `Search-SAT`, in polynomial time

Proof. Let $\varphi(x_1, \ldots, x_n)$ be a CNF formula. Assume an oracle/procedure `SAT`$(\cdot)$ that runs in polynomial time and returns YES iff its input is satisfiable.

Algorithm 1: Search-SAT via SAT oracle (self-reduction)

```
  Input: CNF formula $\varphi(x_1, \ldots, x_n)$
  Output: Satisfying assignment $(x_1, \ldots, x_n)$ or UNSAT
1 if SAT($\varphi$) = NO then
2     output UNSAT;
3 for $i \leftarrow 1$ to $n$ do
4     if SAT($\varphi \wedge x_i$) = YES then
5         set $x_i \leftarrow 1$;   $\varphi \leftarrow \varphi[x_i := 1]$;
6     else
7         set $x_i \leftarrow 0$;   $\varphi \leftarrow \varphi[x_i := 0]$;
8 output $(x_1, \ldots, x_n)$;
```

Correctness. We show by induction on i that at the start of iteration i the current formula φ is satisfiable.

Base case. Step 1 either halts correctly with UNSAT, or proceeds with a satisfiable φ.

Inductive step. Suppose the current φ is satisfiable.

- If $\mathsf{SAT}(\varphi \wedge x_i) = \text{YES}$, then some satisfying assignment sets $x_i = 1$, hence fixing $x_i = 1$ preserves satisfiability.
- If $\mathsf{SAT}(\varphi \wedge x_i) = \text{NO}$, then no satisfying assignment has $x_i = 1$, so every satisfying assignment must have $x_i = 0$. Thus $\varphi \wedge \neg x_i$ is satisfiable, and fixing $x_i = 0$ preserves satisfiability.

After n iterations all variables are fixed and satisfiability has been preserved, yielding a satisfying assignment.

Complexity. There are at most $1 + n$ oracle calls (one initial check and one per variable), each on a formula of size $O(|\varphi|)$. If SAT runs in time $p(|\varphi|)$ for a polynomial p, the overall running time is $O(n \cdot p(|\varphi|) + \text{poly}(|\varphi|))$, hence polynomial.

Conclusion. This is a polynomial-time Cook/Turing reduction from Search-SAT to SAT (self-reducibility of SAT), so if SAT $\in \mathbf{P}$ then Search-SAT $\in \mathbf{P}$. $\square$

Question 6.3.

What kind of Reduction was that?

Answer. We assumed a "subroutine" A that solves `Circ-SAT` Presented an algorithm B that uses A and solves `Search-SAT`. This is known as Cook or Turing Reduction. $\square$

Definition 6.13: Cook Reduction & Turing Reduction

Cook or **Turing reduction**: Given a subroutine A that solves some "problem" Y, B uses A to solve some other problem X. Notation: $X \leq_T Y$.

Comment 6.6

Algorithm B, given access to a black-box/oracle O that decides Y, decides X in polynomial time (notation is B^O decides X).

Comment 6.7

Oracle calls and answers are considered as one unit of computation.

Exercise 6.2

For any language L, $L \leq_T \overline{L}$. Why?

Proof. Given an oracle O for deciding $\overline{L}$, design a poly-time $B^O(x)$ as follows: call $O(x)$ to get a bit b, and then return $\overline{b}$. $\square$

Question 6.4.

Is $L \leq_p \overline{L}$ for all languages $L \in \text{NP}$?

Answer. Not unless $\mathsf{NP} = \mathsf{coNP}$. We show that for any $L \in \text{NP}$, $L \leq_P \texttt{Taut}$ because this would then implies that $\text{NP} \subseteq \text{coNP}$. We have the following relation:

$$L \leq_P \texttt{SAT} \ \& \ \overline{L} \leq_P L \implies \overline{L} \leq_P \texttt{SAT} \implies L \leq_P \overline{\texttt{SAT}} \leq_P \texttt{Taut}$$

as desired. □

Theorem 6.13

Suppose L is any NP language, then there is a Turing reduction from the search version of L.

6.4 Can Randomization Help Computation?

It seems like randomization does help computation, the following are examples:

Example 6.10

Primality Testing (2002): there exists a poly-time randomized algorithm A with uses n^c random coins on an n-bit input such that for all $x \in \{0,1\}^n$,

$$\Pr_{r \leftarrow \{0,1\}^{n^c}}[A(x,r) \text{ is incorrect}] \leq 2^{-n}$$

Example 6.11

Suppose we are given a polynomial $q(x)$ of degree d over $\mathbb{Z}_p$ for a prime p. We want to answer, is $q(x) = 0$. A randomized algorithm is to randomly pick a from $\mathbb{Z}_p$ and return 1 if and only if $q(a) = 0$. For all polynomial q,

$$\Pr_a[A(q) \text{ is incorrect}] \leq \frac{d}{p}$$

To boost the correctness, we can perform the above process 100 times under independently random as. If all of them return 0, return 1; else return 0. In this case:

- If $q = 0$, $\Pr A$ outputs $1 = 1$;
- If $q \neq 0$, $\Pr A$ outputs $1 \leq (d/p)^{100}$;

Example 6.12

Given three matrices A, BC over $\mathbb{Z}_2$, we want to answer whether $AB = C$. The best deterministic algorithm[a] runs in $O(n^{2.377})$. Note that $O(n^2)$ is a lower-bound on any deterministic algorithm running time because it should read the input. We have the following randomized algorithm:

$$\text{Sample a random column vector } x \in \{0,1\}^n \text{ and check } ABx = Cx$$

This has running time $O(n^2)$ because $ABx = A(Bx)$, and

- If $AB = C$, then R always return 1;
- If $AB \neq C$, then R outputs 0 with probability at least 1/2. This is because if $AB \neq C$, let v be the first non-zero row of $AB - C$, then

$$\Pr_x[ABx = Cx] \leq \Pr_x\left[\sum_{i=1}^{n} v_i x_i \mod 2 = 0\right] = \frac{1}{2}$$

[a]as of the year 2025

Lecture 19 - Wednesday, November 12

6.4.1 Probabilistic Turing Machine (PTM)

Definition 6.14: Probabilistic Turing Machine

A **probabilistic Turing machine** is a non-deterministic Turing machine that chooses between the available transitions at each point according to some probability distribution.

Note 6.5

The computation is denoted as $M(x, r)$, where $|r|$ is fixed based on the length of $|x|$.

6.4.2 Class BPP (Bounded Probabilistic Poly Time)

Definition 6.15: Bounded Probabilistic Poly Time (BPP)

We say $L \in \mathsf{BPP}$ if there exists a polynomial-time deterministic TM M and a polynomial p such that for every $x \in \{0,1\}^*$, we have

$$\Pr_{r \leftarrow \{0,1\}^{p(|x|)}}[M(x, r) = L(x)] \geq \frac{2}{3}$$

Comment 6.8

We sometimes refer to such M as probabilistic poly-time (PPT) TMs, signifying that the r part is chosen uniformly at random.

Note 6.6

Let $M(x)$ be a random variable denoting the output of $M(x, r)$ where the randomness value r is picked uniformly at random from the underlying space.

6.4.3 Class RP and coRP

Definition 6.16: RP

We say $L \in \mathsf{RP}$, if there exists a PTT TM M such that

- for all $x \in L$, $\Pr[M(x) = 1] \geq 2/3$;
- for all $x \notin L$, $\Pr[M(x) = 0] = 1$;

Result 6.3

We have $\mathsf{RP} \subseteq \mathsf{NP}$.

Definition 6.17: coRP

We define coRPas the complement of RP. That is, $L \in \mathsf{coRP}$ if and only if $\overline{L} \in \mathsf{RP}$.

Note 6.7

If $L \in \mathsf{coNP}$, then for any $x \in L$, the PPT TM M will always output 1, and if $x \notin L$, it will output 0 with probability at least 2/3.

6.4.4 RP Error Reduction

Question 6.5.

How to decrease the error for RP?

Answer. Suppose we have a PTT TM M such that

- for all $x \in L$, $\Pr[M(x) = 1] \geq 2/3$;
- for all $x \notin L$, $\Pr[M(x) = 0] = 1$;

Then, we can run it k times and output 1 if and only if at least one run outputs 1. Formally speaking, we define $M'(x, r^{(1)}, \dots, r^{(k)})$ as running $M(x, r^{(1)}), \dots, M(x, r^{(k)})$ and output 1 if at least one of them outputs one. then we have

- for all $x \in L$, $\Pr[M(x) = 1] \geq 1 - (1/3)^k$;
- for all $x \notin L$, $\Pr[M(x) = 0] = 1$;

-0.5cm

□

Primality Test: One-sided PPT Given a natural number n, we want to determine in *poly*$(\log n)$ time if n is prime. By Fermat, we know that

$$\boxed{\text{If } n \text{ is a prime, then for all } a \in [n-1],\ a^{n-1} = 1 \pmod{n}.}$$

Therefore, if there exists $a \in [n-1]$ such that $a^{n-1} \neq 1 \pmod{n}$, then this a is a witness to n's compositeness.

First Attempt Here is our first algorithm:

Algorithm 2: Primality Test Attempt 1

```
Input: Natural number $n$
Output: Primality of input $n$
1 Sample $a \leftarrow [n-1]$;
2 if $a^{n-1} \neq 1 \pmod{n}$ then
3     return Composite
4 return Prime
```

Note 6.8

Note that if n is prime, we always output "prime". However, what if the number n is compositive? By Lagrange, we have the following result:

> If there exists $a \in [n-1]$ such that $a^{n-1} \neq 1 \pmod{n}$, then there are at least half of the elements in $[n-1]$, say b, satisfies that $b^{n-1} \neq 1 \pmod{n}$.

Therefore, we output "composite" for such composite numbers with probability at least $1/2$.

Comment 6.9

Such composite numbers n are called Carmichael, but not all compositive numbers are Carmichael. We will later refine this algorithm.

6.4.5 BPP Error Reduction

Question 6.6.

How to decrease the error for BPP.

Answer. Suppose we have a PTT TM M such that

- for all $x \in L$, $\Pr[M(x) = 1] \geq 2/3$;
- for all $x \notin L$, $\Pr[M(x) = 0] = 2/3$;

The error reduction technique we saw for RP doesn't work for BPP. We need to do it another way, and use tail inequalities (e.g., Chernoff bounds) to analyze it. □

Chernoff Bound Let $X_1, \ldots, X_t$ be independent and identically-distributed (i.i.d) Bernouli random variables, where $\Pr[X_1 = 1] = \mu$. Let $X = (X_1 + \cdots + X_t)/t$, then

$$\Pr[|X - \mu| \geq \varepsilon] \leq \frac{2}{\exp(t\varepsilon^2/4)}$$

Theorem 6.14: Error Reduction for BPP

For $L \subseteq \{0,1\}^*$, suppose there exists a PPT TM M and a constant c such that for all x,

$$\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$$

Then, for any constant d, there exists a PPT TM M' such that for all x,

$$\Pr[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}$$

Proof. We first define our new PPT TM M' as: Let

$$m = \left\lceil \frac{4\,n^{2c}\left(n^d+1\right)-1}{2} \right\rceil,$$

where $n = |x|$, and run $M(x)$ $2m+1$ times and output **majority**. Let $y_i = 1$ if the ith trial outputs the correct bit $L(x)$. We have

$$\mu = \Pr[y_i = 1] \;\geq\; \frac{1}{2} + \frac{1}{n^c}.$$

Let $Y = \dfrac{y_1 + \cdots + y_{2m+1}}{2m+1}$, then

$$\Pr[\text{majority is wrong}] \;\leq\; \Pr\big[Y \leq \tfrac{1}{2}\big] \;\leq\; \Pr\big[\,|Y-\mu| \geq \tfrac{1}{n^c}\big] \;\leq\; e^{-\frac{2m+1}{4n^{2c}}} \;\leq\; \frac{1}{2^{n^d}}.$$

due to Chernoff established above. □

Lecture 20 - Monday, November 17

Theorem 6.15: Strong Error Reduction for BPP

Suppose for $\beta > \varepsilon = 1/\text{poly} > 0$

$$x \in L \Rightarrow \Pr[M(x) = 1] > \beta + \varepsilon$$
$$x \notin L \Rightarrow \Pr[M(x) = 1] < \beta - \varepsilon$$

Then we can make the error probability exponentially small

$$\frac{1}{2^{|x|^d}}$$

as in the above theorem.

Proof Attempt: Given x:

1. Compute $\mu := \Pr[M(x) = 1]$. This can be done deterministically in exponential time by enumerating over all randomness values.
2. If μ is closer to $\beta + \varepsilon$ (i.e., $|\mu - (\beta+\varepsilon)| \leq |\mu - (\beta-\varepsilon)|$) output 1, else 0.

This deterministic alg is always correct but runs in exponential time.
Idea: Approximate μ via μ' in PPT within an additive factor $\varepsilon/3$.

- Namely, we want to have: with probability ≈ 1,
$$|\mu' - \mu| \le \frac{\varepsilon}{3}.$$
- Replace μ in Condition (2) above with μ'.
- With probability ≈ 1, we can correctly predict if $x \in L$.

Proof Sketch.
- Approximate
$$\mu := \Pr[M(x) = 1]$$
within an additive factor $\varepsilon/3$ with probability error at most $1/2^{n^d}$. That is, find μ' such that
$$\Pr\Big[|\mu - \mu'| \ge \frac{\varepsilon}{3}\Big] \le \frac{1}{2^{n^d}}.$$
- If μ' is closer to $\beta + \varepsilon$ (i.e.,
$$|\mu' - (\beta + \varepsilon)| \le |\mu' - (\beta - \varepsilon)|),$$
then output 1, else output 0.

To compute such a μ', run $M(x)$
$$m := \frac{36\,(n^d + 1)}{\varepsilon^2}$$
times, obtaining outputs $y_1, \dots, y_m \in \{0, 1\}$, and set
$$\mu' := \frac{y_1 + \cdots + y_m}{m},$$
where y_i is the output of the ith execution. □

6.4.6 BPP is a subset of "non-constructive P"

Theorem 6.16

Any language in BPP can be solved via a non-constructive deterministic poly-time algorithm.

Proof. Reduce the error sufficiently small and argue there exists a choice of randomness (non-constructive part) that works for all strings.
Suppose $\Pr[M(x) \ne L(x)] \le \frac{1}{2^{|x|+1}}$. Then for any $x \in \{0,1\}^n$, at most a $\frac{1}{2^{n+1}}$ fraction of randomness values are bad. Thus, the fraction of randomness values that are bad for some $x \in \{0,1\}^n$ is at most $\frac{1}{2}$. □

Comment 6.10

There is evidence that NP cannot be solved non-constructively in a deterministic way.

Lecture 21 - Wednesday, November 19

6.5 Complexity Class ZPP

Definition 6.18: ZPP

ZPP is the class of languages that can be solved in expected poly time on every input, and always produces the correct output.

Definition 6.19: Ztime($T(n)$)

The class **Ztime($T(n)$)** are languages L for which there exists a PPT TM M that on any input $x \in \{0,1\}^n$ runs in expected time $O(T(n))$ such that if $M(x)$ halts, it outputs the correct bit $x \in^? L$.

Proposition 6.3

We have

$$\mathsf{ZPP} = \bigcup_{c \geq 0} Ztime(n^c)$$

Theorem 6.17

We have

$$\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$$

Proof. We can show $\mathsf{ZPP} \subseteq \mathsf{RP}$ (and similarly $\mathsf{ZPP} \subseteq \mathsf{coRP}$) using the Markov inequality, which states that any nonnegative random variable X satisfies

$$\Pr\big(X \geq k\mathbb{E}[X]\big) \leq \frac{1}{k}.$$

If M runs in expected time $T(n)$, run M for $3T(n)$ steps, and output yes if it halts and says yes; otherwise, say no. How to prove $\mathsf{RP} \cap \mathsf{coRP} \subseteq \mathsf{ZPP}$? □

Lecture 22 - Monday, November 24

6.6 Interractive Proof IP

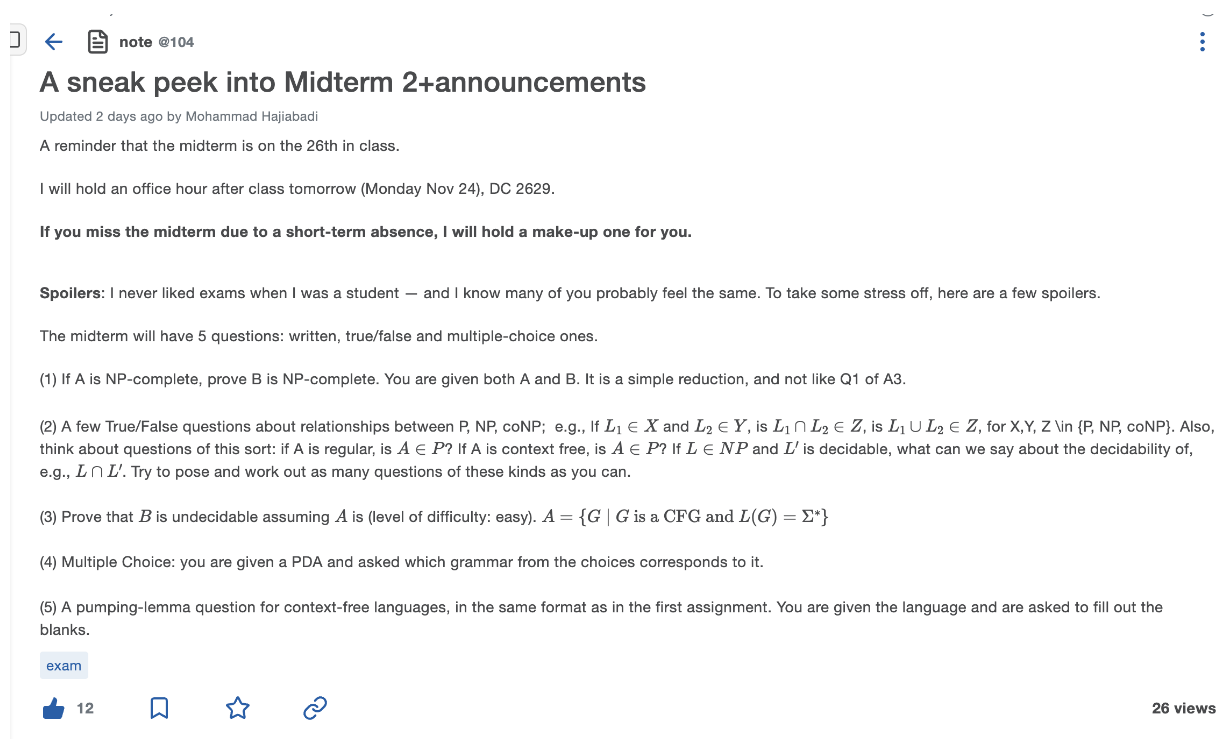
The **Interactive Proof Systems** is characterized by an unbounded prover P and a poly-time verifier V. Let $out_V\langle V, P\rangle$ denote the output bit of the verifier V in an interactive protocol with P. A language L belongs to IP if there exist V, P such that for all Q, w:

$$\begin{aligned} \text{Soundness:} \quad & w \in L \implies \Pr[out_V\langle V, P\rangle = 1] \geq \frac{2}{3}; \\ \text{Completeness:} \quad & w \notin L \implies \Pr[out_V\langle V, Q\rangle = 1] \leq \frac{1}{3}. \end{aligned}$$

Note 6.9

We can design interactive proofs (in fact non-interactive proofs) for all language in NP, by a prover simply sending a witness.

7 Second Midterm Practices


note @104

A sneak peek into Midterm 2+announcements

Updated 2 days ago by Mohammad Hajiabadi

A reminder that the midterm is on the 26th in class.

I will hold an office hour after class tomorrow (Monday Nov 24), DC 2629.

If you miss the midterm due to a short-term absence, I will hold a make-up one for you.

Spoilers: I never liked exams when I was a student — and I know many of you probably feel the same. To take some stress off, here are a few spoilers.

The midterm will have 5 questions: written, true/false and multiple-choice ones.

(1) If A is NP-complete, prove B is NP-complete. You are given both A and B. It is a simple reduction, and not like Q1 of A3.

(2) A few True/False questions about relationships between P, NP, coNP; e.g., If $L_1 \in X$ and $L_2 \in Y$, is $L_1 \cap L_2 \in Z$, is $L_1 \cup L_2 \in Z$, for X,Y, Z \in {P, NP, coNP}. Also, think about questions of this sort: if A is regular, is $A \in P$? If A is context free, is $A \in P$? If $L \in NP$ and L' is decidable, what can we say about the decidability of, e.g., $L \cap L'$. Try to pose and work out as many questions of these kinds as you can.

(3) Prove that B is undecidable assuming A is (level of difficulty: easy). $A = \{G \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$

(4) Multiple Choice: you are given a PDA and asked which grammar from the choices corresponds to it.

(5) A pumping-lemma question for context-free languages, in the same format as in the first assignment. You are given the language and are asked to fill out the blanks.

exam

12 26 views

7.1 NP-Complete

Exercise 7.1

Assuming 3SAT is NP-complete, prove that EXACT-3SAT is NP-complete. Recall

- **3SAT**: Input is a CNF formula φ where each clause has at most 3 literals. Question: is φ satisfiable?
- **EXACT-3SAT**: Input is a CNF formula ψ where each clause has exactly 3 literals. Question: is ψ satisfiable?

Proof. For clauses with three literals keep them as they are. For clauses with two literals, say $x \vee y$, replace it with

$$(x \vee y \vee z) \wedge (x \vee y \vee \neg z)$$

where z is a fresh variable. For clauses with one literal, say x, introduce new variables y and z and replace the clause with

$$(x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z)$$

The rest of the proof is omitted. □

Exercise 7.2

Assuming CLIQUE is NP-complete, prove that INDEPENDENT_SET is NP-complete. Recall

- **CLIQUE**: Input is a graph $G = (V, E)$ and an integer k. Question: does G contain a clique of size at least k?
- **INDEPENDENT_SET**: Input is a graph $G' = (V', E')$ and an integer k'. Question: does G' contain an independent set of size at least k'?

Proof. Just take the graph complement. □

Exercise 7.3

Assuming SUBSET_SUM is NP-complete, prove that PARTITION is NP-complete. Recall

- **SUBSET_SUM**: Input is a multiset of positive integers $S = \{a_1, \dots, a_n\}$ and a target integer T. Question: is there a subset $U \subseteq S$ such that $\sum_{a \in U} a = T$?
- **PARTITION**: Input is a multiset of positive integers $S' = \{b_1, \dots, b_m\}$. Question: can S' be partitioned into two subsets X and $S' \setminus X$ such that

$$\sum_{x \in X} x = \sum_{x \in S' \setminus X} x\,?$$

7.2 True or False

Exercise 7.4

If $L_1, L_2 \in \mathsf{P}$, then $L_1 \cap L_2 \in \mathsf{P}$. **Yes** **No**

Proof. True. □

Exercise 7.5

If $L_1, L_2 \in \mathsf{P}$, then $L_1 \cup L_2 \in \mathsf{P}$. **Yes** **No**

Proof. True. □

Exercise 7.6

If $L \in \mathsf{P}$, then $\overline{L} \in \mathsf{NP}$. **Yes** **No**

Proof. True. because P is closed under complement and $\mathsf{P} \subseteq \mathsf{NP}$. □

Exercise 7.7

If $L \in \mathsf{NP}$ and L' is decidable, then $L \cap L'$ is decidable. **Yes** **No**

Proof. True. Every NP language is decidable (simulate the NP machine deterministically), and the intersection of two decidable languages is decidable. □

Exercise 7.8

If $L \in \mathsf{NP}$ and L' is decidable, then $L \cap L'$ is decidable. **Yes** **No**

Proof. True. Every NP language is decidable (simulate the NP machine deterministically), and the intersection of two decidable languages is decidable. □

Exercise 7.9

If L is undecidable and L' is decidable, then $L \cap L'$ is undecidable. **Yes** **No**

Proof. False. Take $L' = \emptyset$; then $L \cap L' = \emptyset$, which *is* decidable. (Also $L' = \Sigma^*$ gives undecidable again, so both are possible.) □

7.3 Prove B is undecidable assuming A is

Exercise 7.10

Prove that B is undecidable assuming A is. (Level of difficulty: easy.)
$A = ALL_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^* \}$
$B = EQ_{CFG} = \{ \langle G_1, G_2 \rangle \mid G_1, G_2 \text{ are CFGs and } L(G_1) = L(G_2) \}$

Proof. Pick G_2 such that $L(G_2) = \Sigma^*$. □

Exercise 7.11

Prove that B is undecidable assuming A is. (Level of difficulty: medium.)
$A = ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$
$B = REG_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) \text{ is a regular language}\}$.

Exercise 7.12

Prove that B is undecidable assuming A is. (Level of difficulty: medium.)
$A = ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$
$B = AMBIG_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } G \text{ is ambiguous}\}$.

7.4 CFL Pumping Lemma

Exercise 7.13

Let

$$L_2 = \{a^n b^m c^n d^m \mid n, m \geq 0\}.$$

Use the CFL pumping lemma to show L_2 is not context-free. Assume L_2 is context-free. Let p be the pumping length. Choose

$$s = \underline{\hspace{3cm}} \in L_2.$$

Write $s = uvxyz$ with the usual conditions of the lemma. Argue that v and y together affect at most two of the four blocks (a's, b's, c's, d's), and pick

$$i = \underline{\hspace{1.5cm}}$$

so that $uv^ixy^iz \notin L_2$, a contradiction.

Exercise 7.14

Let

$$L_4 = \{ww \mid w \in \{0,1\}^*\}.$$

Use the CFL pumping lemma to show that L_4 is not context-free. Assume L_4 is context-free and let p be the pumping length from the lemma. Consider the string

$$s = \underline{\hspace{4.5cm}} \in L_4$$

of length at least p where the first half and second half are carefully chosen. For every decomposition $s = uvxyz$ with $|vxy| \leq p$ and $|vy| > 0$, explain why v and y must lie entirely within the first half of s, entirely within the second half, or overlap the middle boundary. In each possible case, select a value

$$i = \underline{\hspace{1.5cm}}$$

and show that uv^ixy^iz is no longer of the form ww, contradicting the lemma.

Index